

TS. VÕ TRUNG HÙNG



# LẬP TRÌNH TRỰC QUAN



Đà Nẵng, 01-2008



# LỜI NÓI ĐẦU

Lịch sử phát triển của Tin học luôn gắn liền với việc tìm kiếm các phương pháp lập trình để giúp cho người sử dụng triển khai các ứng dụng một cách dễ dàng, nhanh chóng và hiệu quả.

Như chúng ta đã biết, mỗi loại máy tính (sử dụng loại CPU – Central Processing Unit xác định) chỉ có thể hiểu và thực hiện trực tiếp được các lệnh cũng như chương trình theo một loại ngôn ngữ dành riêng được gọi là ngôn ngữ máy. Tuy nhiên, nếu triển khai các ứng dụng trong thực tế mà phải viết chương trình trực tiếp bằng ngôn ngữ máy thì sẽ rất phức tạp, đòi hỏi thời gian và công sức rất lớn, nhiều khi không thể thực hiện được. Vì vậy, người ta tìm cách xây dựng một ngôn ngữ lập trình riêng gần với các ngôn ngữ tự nhiên, thuận lợi cho việc triển khai các ứng dụng. Khi thực hiện các chương trình bằng ngôn ngữ này phải qua một bước dịch chương trình đó sang ngôn ngữ máy để nó có thể thực hiện. Từ trước đến nay có rất nhiều ngôn ngữ lập trình được ra đời và phục vụ đắc lực cho việc triển khai các ứng dụng trên máy tính.

Trong giai đoạn đầu, các ngôn ngữ lập trình tuy dễ sử dụng hơn ngôn ngữ máy nhưng rất khó với các lập trình viên vì chưa đủ mạnh để dễ dàng triển khai các thuật toán. Chương trình chưa có tính cấu trúc chặt chẽ về mặt dữ liệu cũng như tổ chức chương trình. Vì vậy, việc triển khai các ứng dụng trong thực tế bằng các ngôn ngữ lập trình này là rất khó khăn.

Giai đoạn 2 là thời kỳ của các ngôn ngữ lập trình có cấu trúc. Các ngôn ngữ lập trình này có đặc điểm là có tính cấu trúc chặt chẽ về mặt dữ liệu và tổ chức chương trình. Một loạt các ngôn ngữ lập trình có cấu trúc ra đời và được sử dụng rộng rãi như : PASCAL, C, BASIC...

Giai đoạn 3 là thời kỳ của lập trình hướng đối tượng và phương pháp lập trình có bước biến đổi mạnh. Trong các ngôn ngữ lập trình có cấu trúc thì một ứng dụng bao gồm hai thành phần riêng là dữ liệu và chương trình. Tuy chúng có quan hệ chặt chẽ nhưng là hai đối tượng riêng biệt. Trong phương pháp lập trình hướng đối tượng thì mỗi một đối tượng lập trình sẽ bao hàm cả dữ liệu và phương thức hành động trên dữ liệu đó. Vì vậy, việc lập trình sẽ đơn giản và mang tính kế thừa cao, tiết kiệm được thời gian lập trình.

Tuy nhiên, với các phương pháp lập trình trên đều đòi hỏi lập trình viên phải nhớ rất nhiều câu lệnh với mỗi lệnh có một cú pháp và tác dụng riêng, khi viết chương trình phải tự lắp nối các lệnh để có một chương trình giải quyết từng bài toán riêng biệt.

Trong xu hướng phát triển mạnh mẽ hiện nay của tin học, số người sử dụng máy tính tăng lên rất nhanh và máy tính được sử dụng trong hầu hết các lĩnh vực của đời sống nên đòi hỏi các ngôn ngữ lập trình cũng phải đơn giản, dễ sử dụng và mang tính đại chúng cao. Chính vì vậy phương pháp lập trình trực quan ra đời. Đặc điểm của các ngôn ngữ lập trình trực quan là dễ sử dụng, triển khai các ứng dụng một cách nhanh chóng.

Đặc điểm nổi bật của phương pháp lập trình trực quan là :

- Cho phép xây dựng chương trình theo hướng sự kiện (Event-Driven Programming, nghĩa là một chương trình ứng dụng được viết theo kiểu này đáp ứng dựa theo tình huống xảy ra lúc thực hiện chương trình. Tình huống này bao gồm người sử dụng ấn một phím tương ứng, chọn lựa một nút lệnh hoặc gọi một lệnh từ một ứng dụng khác chạy song song cùng lúc.
- Người lập trình trực tiếp tạo ra các khung giao diện (interface), ứng dụng thông qua các thao tác trên màn hình dựa vào các đối tượng (object) như hộp hội thoại hoặc nút điều

hiển (control button), những đối tượng này mang các thuộc tính (properties) riêng biệt như : màu sắc, Font chữ.. mà ta chỉ cần chọn lựa trên một danh sách cho sẵn.

- Khi dùng các ngôn ngữ lập trình trực quan ta rất ít khi phải tự viết các lệnh, tổ chức chương trình... một cách rắc rối mà chỉ cần khai báo việc gì cần làm khi một tình huống xuất hiện.
- Máy tính sẽ dựa vào phần thiết kế và khai báo của lập trình viên để tự động tạo lập chương trình.

Như vậy với kỹ thuật lập trình trực quan, lập trình viên giống như một nhà thiết kế, tổ chức để tạo ra các biểu mẫu, đề nghị các công việc cần thực hiện và máy tính sẽ dựa vào đó để xây dựng chương trình. Hiện nay các ngôn ngữ lập trình, hệ quản trị cơ sở dữ liệu theo hướng trực quan thường dùng như : Visual Basic, Visual Foxpro, Visual C, Delphi...

Cuốn sách này được viết dựa trên cơ sở của giáo trình “Lập trình trực quan” được giảng dạy tại Khoa Công nghệ Thông tin, Trường Đại học Bách Khoa Đà Nẵng từ năm 1994 đến nay. Nội dung của cuốn sách gồm ba phần chính: Phần thứ nhất (tương ứng với chương 1) giới thiệu tổng quan về ngôn ngữ lập trình trực quan; Phần thứ hai (tương ứng với chương 2) giới thiệu về lập trình trực quan với MS Access, đây là công cụ mạnh để quản trị các cơ sở dữ liệu và phát triển các phần mềm quản lý; Phần thứ ba (từ các chương 3 đến 10) tập trung giới thiệu cách thức lập trình với ngôn ngữ Visual Basic, đây là ngôn ngữ lập trình trực quan hiện đại và cung cấp cho người sử dụng những công cụ mạnh để thiết kế giao diện, kết nối đến cơ sở dữ liệu, xây dựng các hiệu ứng đồ họa...

Chúng tôi hy vọng là cuốn sách này sẽ giúp ích nhiều cho các sinh viên tại các trường đại học, cao đẳng; các học viên tại các cơ sở đào tạo lập trình viên; các lập trình viên... trong việc tìm hiểu, khám phá các ngôn ngữ lập trình trực quan. Đặc biệt, cuốn sách sẽ giúp các bạn làm chủ được MS Access và Visual Basic để phát triển các ứng dụng và trên cơ sở đó có thể tự nghiên cứu để làm chủ các ngôn ngữ lập trình trực quan khác.

# CHƯƠNG 1. LẬP TRÌNH TRỰC QUAN

## I. Giới thiệu

Từ những bức tranh đầu tiên tìm thấy trong các hang động thời tiền sử cho đến chữ tượng hình rồi đến những bức tranh của *Campbell's soup cans*, nhân loại đã trải qua một thời gian dài sử dụng hình ảnh như là công cụ truyền thông hữu hiệu. Lập trình trực quan ra đời để trả lời câu hỏi tại sao chúng ta không thử giao tiếp với máy tính bằng chế độ đồ họa (graphic mode) mà bằng chế độ văn bản (text mode)? Chúng ta có thể sử dụng máy tính để giao tiếp qua hình ảnh được không và tại sao không làm việc với máy tính một cách trực quan sinh động khi giải quyết một số bài toán đặt ra trong thực tế? Một cách hiển nhiên, những kết quả đạt được từ các ngôn ngữ lập trình trực quan (VPL - Visual Programming Languages) đã trả lời rằng hai vấn đề nêu trên là hoàn toàn có thể thực hiện được.

Những câu hỏi trên là động lực để người ta tiến hành các nghiên cứu để tạo ra các ngôn ngữ VPL. Trước hết, nhiều người suy nghĩ và nhớ lại những vấn đề liên quan đến khả năng biểu đạt của hình ảnh. Người ta liên hệ đến thế giới thực theo cách mà đồ họa đã có và sử dụng hình ảnh như thành tố đầu tiên của tư duy sáng tạo. Thêm vào đó, các ngôn ngữ lập trình trong chế độ văn bản (Textual Programming Languages) đã bộc lộ những hạn chế, đặc biệt là khi người sử dụng phải học rất nhiều để sử dụng các sản phẩm phần mềm được tạo ra bởi chúng.

Việc giảm thiểu hoặc xóa bỏ hoàn toàn khoảng cách từ ý tưởng đến phát triển ứng dụng rồi học tập để sử dụng là rất cần thiết. Hơn nữa, sự đa dạng của các ứng dụng bao gồm khoa học hình dung, sự tương tác giữa sáng tạo và mô phỏng đã góp phần đặc biệt tạo ra các phương pháp phát triển trực quan.

Trong chương đầu tiên này, chúng tôi tập trung giới thiệu những kết quả ban đầu trong lĩnh vực phát triển các ngôn ngữ lập trình trực quan nhằm trả lời những vấn đề đã đặt ra ở trên. Chúng tôi bắt đầu bằng việc giới thiệu đôi nét về lịch sử và những công việc/kết quả ban đầu đã được sử dụng làm cơ sở để tiến hành các nghiên cứu hiện đại hơn trong lĩnh vực này. Chúng tôi cũng sẽ trình bày việc phân loại và những quan tâm về mặt lý thuyết chính của các ngôn ngữ trực quan. Phần cuối của chương này, chúng tôi giới thiệu một số ngôn ngữ lập trình trực quan và một số ví dụ về chúng cũng như những định hướng nghiên cứu thuộc lĩnh vực này trong tương lai.

## II. Lịch sử của các ngôn ngữ lập trình trực quan

Lĩnh vực lập trình trực quan là sự kết hợp giữa đồ họa máy tính, ngôn ngữ lập trình và tương tác người máy. Nó được tạo ra từ nhiều nghiên cứu độc lập và có nhiều công trình nghiên cứu được xem là tiên phong trong lĩnh vực này.

Chúng tôi muốn giới thiệu một nghiên cứu đầu tiên trong lĩnh vực này là hệ thống Sketchpad của Ivan Sutherland vào năm 1963. Sketchpad đã được thiết kế và thực hiện trên máy tính TX-2 tại MIT và nó được xem là ứng dụng đồ họa đầu tiên trên máy tính. Hệ thống cho phép người sử dụng làm việc với một bút vẽ để tạo ra các hình ảnh trong không gian 2 chiều (2D) đơn giản như đường thẳng, đường tròn các xử lý như sao chép và các phép biến đổi hình học khác trên các hình vẽ này.

Giao diện đồ hoạ và những hỗ trợ của nó dành cho người sử dụng được xem là bước đột phá và có đóng góp quan trọng để tạo nên những ngôn ngữ lập trình trực quan sau này. Bằng cách định nghĩa các ràng buộc thích hợp, người sử dụng có thể phát triển các cấu trúc như các liên kết cơ học phức tạp và tiếp đến cho chúng chuyển động như trong hệ thống thời gian thực. Chúng ta có thể thấy ý tưởng sử dụng các ràng buộc được đặc tả trực quan cũng như cách lập trình hướng đối tượng được dùng lại rất nhiều trong các ngôn ngữ VPL sau này. Em trai của Ivan Sutherland là William cũng đã có những đóng góp quan trọng trong lĩnh vực lập trình trực quan. Từ năm 1965, ông ta đã sử dụng máy tính TX-2 để phát triển một ngôn ngữ đặc tả dòng dữ liệu trực quan đơn giản. Hệ thống cho phép người sử dụng khởi tạo, dò lỗi và thực thi các biểu đồ dòng dữ liệu trong môi trường trực quan.

Cột mốc kế tiếp trong việc phát triển các ngôn ngữ VPL là năm 1975 với việc công bố luận án tiến sĩ của David Canfield Smith với tiêu đề “Pygmalion: A Creative Programming Environment”. Công việc của Smith đánh dấu sự khởi đầu của một loạt các nghiên cứu trong lĩnh vực này cho đến ngày hôm nay. Ví dụ, Pygmalion thể hiện một mô hình lập trình dựa trên các biểu tượng (icon), trong đó người sử dụng có thể khởi tạo, sửa đổi và liên kết chúng đến những đối tượng hình ảnh nhỏ gọi là các icons và có thể định nghĩa các thuộc tính để thực hiện các tính toán. Những kết quả của Smith cùng nhiều thành tựu nghiên cứu khác sau này đã góp phần hình thành nên tiêu chuẩn về lý thuyết biểu tượng (icon theory). Nhiều ngôn ngữ VPL hiện đại sử dụng cách tiếp cận dựa trên lý thuyết biểu tượng này. Pygmalion cũng đã sử dụng khái niệm của lập trình bằng ví dụ, ở đó người sử dụng thấy được làm thế nào mà hệ thống thực hiện một công việc trong một trường hợp đặc biệt và hệ thống sử dụng thông tin để phát sinh tự động một chương trình để nó thực hiện công việc trong trường hợp tổng quát nhất. Trong hệ thống của Smith, người sử dụng thiết lập môi trường để nhớ chế độ (“remember” mode), thực hiện việc tính toán, tắt chế độ nhớ, nhận các kết quả của một chương trình trong một tập con đơn giản giống như tập con của Smalltalk, để thực hiện việc tính toán trên một đầu vào bất kỳ.

### **III. Phân loại các ngôn ngữ lập trình trực quan**

Khi lĩnh vực VPL đã chín muồi, ngày càng nhiều những nghiên cứu hướng đến việc sáng tạo mạnh mẽ hơn cũng như những tiêu chuẩn để phân loại những công việc trong lĩnh vực này. Một hệ thống phân loại không chỉ giúp các nhà nghiên cứu trong việc xác định các công việc liên quan mà còn cung cấp một ranh giới để so sánh và đánh giá các hệ thống khác nhau. Một số tên tuổi có những đóng góp quan trọng trong lĩnh vực này như Chang, Shu, và Burnett. Họ đã phân loại và đưa ra những định nghĩa để xác định đặc trưng các nhóm chính của VPL.

Sau đây là sự phân loại các ngôn ngữ lập trình trực quan chính:

- Các ngôn ngữ trực quan thuần túy
- Những hệ thống lai giữa trực quan và văn bản (text)
- Những hệ thống lập trình bằng ví dụ
- Những hệ thống ràng buộc đối tượng
- Những hệ thống được xây dựng dựa trên các biểu mẫu (form)

Lưu ý rằng việc phân loại trên là không loại trừ lẫn nhau. Do đó, có nhiều ngôn ngữ có thể thuộc nhóm này nhưng cũng có thể thuộc nhóm khác.

Chỉ riêng nhóm các ngôn ngữ trực quan thuần túy (Purely Visual Languages) là quan trọng nhất. Các ngôn ngữ này được đặc trưng bởi việc dựa hoàn toàn trên các kỹ thuật trực quan thông qua tiến trình lập trình. Người lập trình vận dụng các biểu tượng hoặc các sự biểu diễn đồ họa khác để khởi tạo một chương trình rồi tiếp theo sẽ gỡ lỗi và thực thi trong cùng một môi trường trực quan. Chương trình là được biên dịch trực tiếp từ sự trình bày trực quan của nó và không bao giờ dịch đến một ngôn ngữ trung gian dựa trên chế độ văn bản (interim text-based language). Những ví dụ của các hệ thống thuần túy trực quan như VIPR, Prograph và PICT. Trong nhiều tài liệu thuộc lĩnh vực này, sự phân loại này là đi xa hơn đến việc phân loại các nhóm con như nhóm ngôn ngữ biểu tượng (iconic languages) và phi biểu tượng (non-iconic languages), ngôn ngữ hướng đối tượng (object-oriented), ngôn ngữ lập trình hàm (functional programming), và ngôn ngữ mệnh lệnh (imperative languages). Tuy nhiên, mục đích của chúng ta là nhắm đến các nhóm chính để làm nổi bật sự khác nhau giữa những ngôn ngữ lập trình hướng trực quan (visually-oriented) và các ngôn ngữ VPL khác.

Một nhóm con quan trọng của các ngôn ngữ VPL là sự móc nối cả hai phần tử trực quan và văn bản (text). Những hệ thống lai này bao gồm cả hai yếu tố, trong đó chương trình được khởi tạo một cách trực quan và sau đó dịch đến ngôn ngữ lớp dưới sử dụng chế độ văn bản và những hệ thống bao gồm việc sử dụng các phần tử đồ họa trong một ngôn ngữ theo chế độ văn bản. Những ví dụ cho loại ngôn ngữ kiểu này là Rehearsal World và Work của Erwig. Trong những hệ thống cũ, người sử dụng huấn luyện hệ thống để giải quyết một vấn đề đặc biệt bởi việc vận dụng các nhân tố đồ họa (graphical “actors”) và tiếp đến hệ thống phát sinh một chương trình Smalltalk. Smalltalk là một ngôn ngữ lập trình và môi trường lập trình bậc cao, xem các quá trình tính toán như là những đối tượng gửi thông báo truyền tin lẫn nhau. Ngôn ngữ này do Alan Kay và các cộng sự tại Trung tâm Nghiên cứu Palo Alto (PARC) của hãng Xerox Corporation xây dựng nên. SmallTalk là một ngôn ngữ khai báo, nó khuyến khích các lập trình viên định nghĩa các đối tượng theo các thuật ngữ liên quan đến ứng dụng dự kiến. Đây là một ngôn ngữ rất mạnh vì có thể xây dựng các đối tượng rất dễ dàng. Vì ngôn ngữ này đòi hỏi bộ nhớ lớn để tạo ra các chương trình chạy nhanh và hiệu quả, nên các lập trình viên chuyên nghiệp vẫn còn ưa thích các ngôn ngữ như C và hợp ngữ. Tuy nhiên, SmallTalk đã truyền cảm hứng để các nhà sản xuất phần mềm cho ra đời HyperTalk, một ngôn ngữ lập trình ứng dụng được cung cấp kèm với tất cả các máy Macintosh bán ra từ 1987. Trong chiếc áo mới của mình, SmallTalk đã hoàn thiện mục tiêu là làm cho việc lập trình máy tính trở nên dễ dàng hơn; hàng chục ngàn người dùng Macintosh đã học được cách lập trình theo HyperTalk.

Ngoài hai nhóm chính này, có nhiều ngôn ngữ VPL được xếp vào các nhóm con. Ví dụ, một số các ngôn ngữ VPL đi theo hướng của Pygmalion bằng cách cho phép người sử dụng khởi tạo và vận dụng các đối tượng đồ họa và ở đó họ “dạy” cho hệ thống biết cách làm thế nào để thực hiện một nhiệm vụ cụ thể được giao. Ngôn ngữ Rehearsal World được nói ở phần trên, được xếp vào nhóm các ngôn ngữ lập trình bằng ví dụ. Một vài ngôn ngữ VPL có thể lần theo dấu vết của nó, một phần như vậy được vận dụng trong các ràng buộc của Sutherland được sử dụng trong Sketchpad. Những hệ thống hướng ràng buộc này (constraint-oriented systems) là đặc biệt phổ biến trong những mô hình lập trình mà các đối tượng vật lý giống như các đối tượng trong môi trường trực quan. Nó là đối tượng để ràng buộc thiết kế để bắt chước cách ứng xử của các qui luật tự nhiên. Những hệ thống hướng ràng buộc cũng được ứng dụng để phát triển các giao diện đồ họa cho người dùng. Hai ngôn ngữ Thinglab và ARK là những mô phỏng đầu tiên của ngôn ngữ VPL. Đây là ví dụ điển hình nhất cho lớp các ngôn ngữ hướng ràng buộc. Một số ít các ngôn ngữ VPL đã vay mượn sự hiển thị và phép lập trình ẩn dụ từ các bảng tính. Những ngôn ngữ này có thể được xếp vào loại các ngôn ngữ VPL được xây dựng trên các biểu mẫu (form-based VPLs). Chúng trình bày việc lập trình như là sự thay

đổi một nhóm các ô có mối liên hệ với nhau trên toàn bộ thời gian và thường cho phép người lập trình hiển thị việc thực thi của một chương trình như một chuỗi các tình trạng diễn ra trên các ô trong tiến trình thực hiện theo thời gian. Forms/3 là hiện thân cho tổ tiên thuộc kiểu ngôn ngữ VPL này và nó sẽ được mô tả rõ hơn ở bên dưới. Cũng quan trọng khi lưu ý rằng trong mỗi một loại đã được đề cập ở trên, chúng ta có thể tìm các ví dụ của cả các ngôn ngữ VPL đa năng (general-purpose VPL) và những ngôn ngữ được thiết kế cho các ứng dụng thuộc một lĩnh vực đặc biệt.

Phạm vi của lập trình trực quan đã được mở rộng trong những năm gần đây. Sự phát triển liên tục và mạnh mẽ của các ngôn ngữ lập trình trong các thể loại được nêu ở trên đã dẫn đến một số công việc mà nó đã được quan tâm từ đầu nhưng không được xem là hình mẫu của lập trình trực quan. Những đứa con mô côi của VPL, theo cách nói này, gồm giải thuật của các hệ thống hoạt hình như là BALSA, nó cung cấp sự hiển thị đồ họa tương tác của các chương trình và các công cụ phát triển giao diện người dùng đồ họa, như chúng đã cung cấp với nhiều trình biên dịch hiện đại như Microsoft Visual C++. Cả hai kiểu của hệ thống nhất định bao gồm các thành phần trực quan nhưng chúng là những ứng dụng mang nặng tính đồ họa hơn là bộ phát sinh mẫu của những ngôn ngữ lập trình hiện tại.

#### **IV. Lý thuyết của các ngôn ngữ lập trình trực quan**

Trong phần này, chúng ta xem xét những lý thuyết tiên tiến trong lĩnh vực các ngôn ngữ lập trình trực quan và chủ yếu là những kết quả từ rất sớm liên quan đến những nghiên cứu của S.-K. Chang trên lý thuyết tổng quát về biểu tượng. Trước khi đi sâu tìm hiểu những vấn đề này chúng ta làm quen với các định nghĩa sau:

##### **icon (biểu tượng, hình tượng)**

Một đối tượng với sự trình bày kép của một phần lô-gíc (ý nghĩa) và một phần vật lý (hình ảnh).

##### **iconic system (hệ thống có tính chất biểu tượng)**

Một tập hợp có cấu trúc của những biểu tượng có liên quan.

##### **iconic sentence (câu biểu tượng/trực quan)**

Một sự sắp xếp không gian của các biểu tượng từ hệ thống biểu tượng.

##### **visual language (ngôn ngữ trực quan)**

Một tập các câu trực quan được cấu trúc với cú pháp và ngữ nghĩa cho trước.

##### **syntactic analysis (phân tích cú pháp)**

Một sự phân tích của một câu trực quan để xác lập cấu trúc cơ sở.

##### **semantic analysis (phân tích ngữ nghĩa)**

Một sự phân tích của một câu trực quan để xác định ngữ nghĩa cơ sở.

Trong phần này, chúng ta giới hạn sự thảo luận liên quan đến các ngôn ngữ trực quan 2 chiều.

#### **1. Đặc tả hình thức của ngôn ngữ lập trình trực quan**

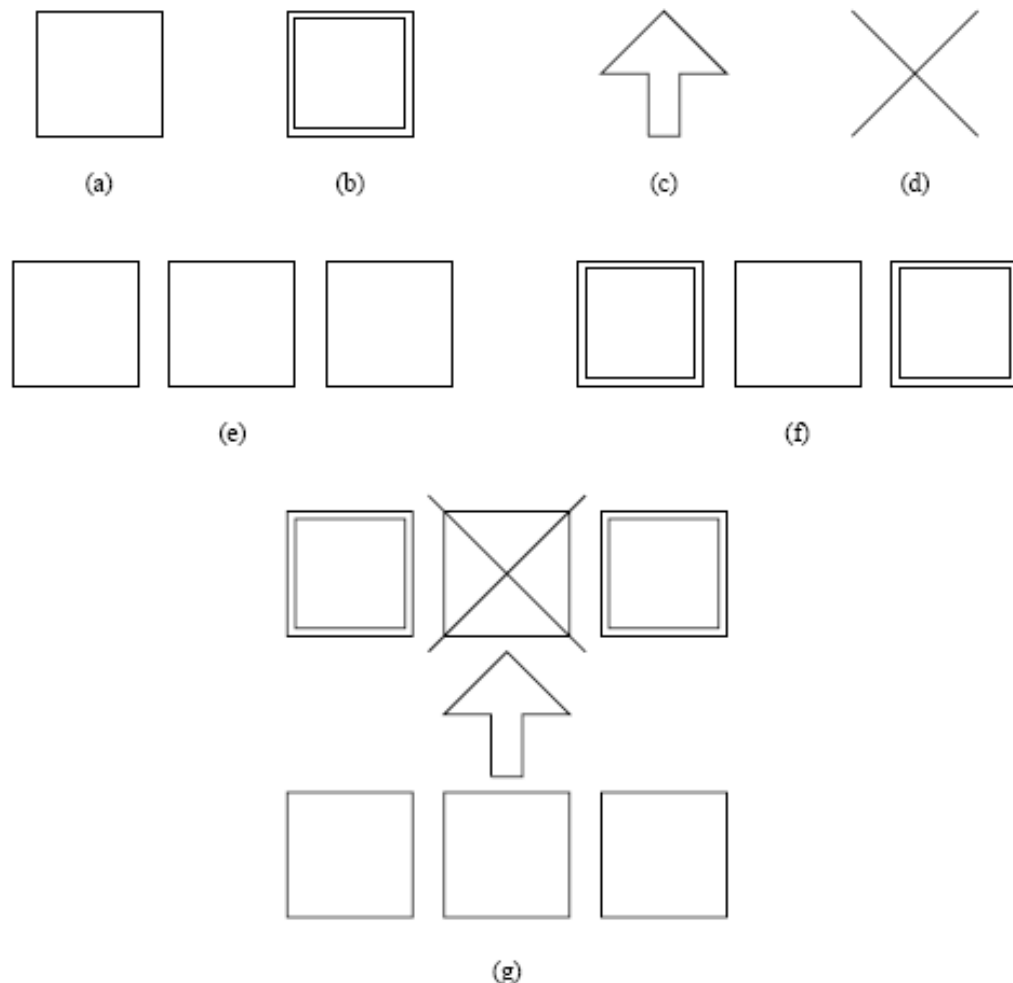
Một sự sắp xếp không gian của các biểu tượng sẽ tạo thành một câu trực quan. Nó là bản sao hai chiều của sự sắp xếp một chiều của các dấu hiệu trong các ngôn ngữ lập trình qui ước



(trong chế độ text). Trong những ngôn ngữ này, một chương trình là được diễn giải như một xâu ký tự trong đó ký hiệu kết thúc là móc nối đến mẫu một câu mà cấu trúc và ngữ nghĩa của nó là được khám phá bởi sự phân tích cú pháp và ngữ nghĩa theo thứ tự định sẵn. Theo cách đó, qui tắc xây dựng là tiềm ẩn trong ngôn ngữ và không cần giải thích như một phần của ngôn ngữ đặc tả. Ngược lại, trong những ngôn ngữ lập trình trực quan chúng ta phân biệt 3 nguyên tắc để sắp xếp các biểu tượng: sự móc nối theo chiều ngang (biểu hiện bởi ký hiệu &), móc nối theo chiều đứng (biểu hiện bởi ký hiệu ^) và không gian bao phủ (biểu hiện bởi ký hiệu +).

Trong việc định dạng các ngôn ngữ lập trình trực quan, nó là lựa chọn để phân biệt các biểu tượng tiến trình từ các biểu tượng đối tượng. Sự tính toán các biểu thức trước, sau đó có thể phân nhỏ đến các biểu tượng đối tượng sơ cấp và các biểu tượng đối tượng phức hợp. Những biểu tượng đối tượng cơ sở cho biết các đối tượng sơ cấp trong ngôn ngữ, nhưng ngược lại các đối tượng phức hợp cho biết các đối tượng được định dạng bởi một sự sắp xếp không gian của các biểu tượng đối tượng sơ cấp.

Tóm lại, thuật ngữ biểu tượng sơ cấp (elementary icons) là đề tham chiếu đến cả hai loại: biểu tượng tiến trình và biểu tượng đối tượng sơ cấp và biểu hiện bởi những biểu tượng của chúng đó là nguyên gốc trong ngôn ngữ. Từ một bức tranh (hoặc biểu tượng trong trường hợp này) là có hàng ngàn cách biểu đạt, chúng ta cố gắng minh họa tất cả những ý niệm trên đây trong hình vẽ sau:



Hình 1. Hệ thống biểu tượng Heidelberg

Hình vẽ này giới thiệu một vài biểu tượng từ bộ biểu tượng Heidelberg. Những biểu tượng sơ cấp gồm (a) là một ký tự và (b) là một ký tự được chọn; các biểu tượng tiến trình: (c) là hoạt động chèn và (d) là hoạt động xoá; các biểu tượng phức hợp: (e) là một xâu (kết hợp các ký tự) và (f) là xâu được lựa chọn; (g) câu trực quan biểu thị sự thay thế của một xâu con trong một xâu nào đó.

Một môn ngữ lập trình trực quan là được đặc tả bởi một bộ ba  $(ID, G0, B)$ , ở đây  $ID$  là từ điển biểu tượng,  $G0$  là một ngữ pháp và  $B$  là một cơ sở tri thức cho một lĩnh vực đặc biệt. Từ điển biểu tượng là một tập hợp các biểu tượng tổng quát và mỗi một phần tử là được trình bày bởi một cặp  $(X_m, X_i)$ , với phần lô-gíc  $X_m$  (nghĩa) và phần vật lý  $X_i$  (hình ảnh). Ngữ pháp  $G0$  chỉ định cách thức làm thế nào để những biểu tượng đối tượng phức hợp có thể được xây dựng từ các biểu tượng sơ cấp bằng việc sử dụng những toán tử sắp xếp không gian.

Cơ sở tri thức  $B$  chứa những thông tin của một lĩnh vực đặc biệt cần thiết cho việc xây dựng ngữ nghĩa của một câu trực quan cho trước. Nó chứa thông tin về các tên sự kiện, những quan hệ thuộc về khái niệm, tên của các đối tượng kết quả và những tham chiếu đến các đối tượng kết quả.

## **2. Phân tích các ngôn ngữ lập trình trực quan**

Như đã thảo luận ở trên, các câu trực quan là đã được xây dựng từ các biểu tượng sơ cấp bằng cách sử dụng các toán tử liên quan đến biểu tượng. Việc phân tích cú pháp của các câu trực quan là được xây dựng trên cơ sở của một số cách tiếp cận của Chang. Ở đây, chúng ta giới thiệu một danh sách không gian của các cách tiếp cận này.

### **Picture-processing grammars (ngữ pháp xử lý ảnh)**

Nguồn gốc của việc thiết kế là phân tích những ảnh số trên một lưới vuông, ở đó những ngữ pháp là được xây dựng trên cơ sở thực tế là những bức ảnh số là sự kết hợp của các điểm ảnh. Ngữ pháp này khám phá ra cấu trúc của câu trực quan bởi trật tự của các điểm ảnh riêng lẻ đến các phần tử trực quan có thể nhận diện được (đường thẳng, hộp, cung tròn...). Cách tiếp cận này có ích khi một hệ thống biểu tượng cần để nhận dạng các biểu tượng với một mức độ nào đó của lỗi có thể chấp nhận được (ví dụ các chữ số viết tay).

### **Precedence grammars (ngữ pháp thứ tự)**

Ngữ pháp phân tích không gian này có thể được sử dụng dành cho việc phân tích các biểu thức toán học trong không gian 2 chiều và phân tích các trang in. Ngữ pháp thứ tự là phù hợp hơn để phân tích cú pháp của các câu trực quan được xây dựng từ các biểu tượng sơ cấp các toán tử biểu tượng. Cây phân tích là được xây dựng bởi việc so sánh thứ tự của các toán tử trong một mẫu và tập con được chia của mẫu đến một hoặc nhiều mẫu con.

### **Context-free and context-dependent grammars (ngữ pháp phụ thuộc ngữ cảnh và ngữ pháp phi ngữ cảnh)**

Những ngữ pháp này được dùng để xác định tổ hợp của các câu trực quan trong việc sử dụng những hình thức quen thuộc và nhiều phương pháp tiêu chuẩn của sự phân tích như những ngữ pháp phù hợp.

### **Graph grammars (ngữ pháp đồ thị)**

Đây là phương thức mạnh nhất (mặc dù kém hiệu quả) để đặc tả các ngôn ngữ trực quan. Những hình thức này cung cấp nhiều ý nghĩa nhất để thiết lập những quan hệ ngữ cảnh và nhiều nghiên cứu hiện tại đã được đầu tư để tạo ra những phân tích có thể tính toán được với ngữ pháp đồ thị.

Một cây phân tích được sản sinh bởi một trong những phương pháp phân tích trên là những phân tích sử dụng cách tiếp cận truyền thống để phân tích ngữ nghĩa (nghĩa là ngữ pháp tượng trưng hoặc cây tính toán...).

## **V. Những vấn đề của ngôn ngữ trực quan**

Chúng ta thảo luận một vài vấn đề chung về ngôn ngữ trực quan. Những vấn đề này là thích hợp nhất với các ngôn ngữ trực quan hướng mục đích (phù hợp để sản sinh chương trình thực thi trong kích cỡ hợp lý), nhưng những vấn đề nào cũng sẽ có liên quan đến những ngôn ngữ thuộc một lĩnh vực đặc biệt (được thiết kế để phục vụ một lĩnh vực đặc biệt như là công nghệ phần mềm hoặc khoa học hiển thị).

### **1. Control Flow (luồng điều khiển)**

Tương tự như những ngôn ngữ lập trình thông thường, những ngôn ngữ trực quan gắn liền với hai khái niệm về luồng điều khiển trong các chương trình là: mệnh lệnh và khai báo.

Với cách tiếp cận mệnh lệnh, một ngôn ngữ lập trình trực quan dựa trên một hoặc nhiều luồng điều khiển hoặc biểu đồ luồng dữ liệu mà ở đó nó cho biết làm thế nào xâu chuỗi các luồng điều khiển thông qua chương trình. Một ưu thế đặc biệt của cách tiếp cận này là nó cung cấp một sự trình bày trực quan hiệu quả theo kiểu song song. Một bất tiện của phương pháp này là một người lập trình phải lưu giữ dấu vết của chuỗi thứ tự nào đó của dãy các hoạt động đã làm thay đổi tình trạng của chương trình, đây là điều không phải là một đặc điểm được mong đợi của một hệ thống (đặc biệt là nếu nó đã được thiết kế để cung cấp cho những người mới học việc).

Một trong số các ngữ nghĩa mệnh lệnh của luồng điều khiển là để sử dụng một kiểu có thể khai báo của lập trình. Với cách tiếp cận này, người ta chỉ lo lắng mỗi việc là những tính toán nào là sẽ được thực hiện và không cần biết làm thế nào những hoạt động hiện tại được tiến hành. Việc thay đổi trạng thái nhất định là đã được đề phòng bởi việc sử dụng chỉ một sự phân công: người lập trình khởi tạo một đối tượng mới bởi việc sao chép một trạng thái của đối tượng đang tồn tại. Cũng vậy, thay vì đặc tả một dãy các thay đổi trạng thái, người lập trình định nghĩa những hoạt động bởi việc đặc tả những phụ thuộc đối tượng. Ví dụ, nếu người lập trình định nghĩa  $Y$  là bằng  $X+1$ , thì rõ ràng  $Y$  được tính toán bằng cách sử dụng đối tượng  $X$ , cho phép hệ thống suy ra rằng giá trị của  $X$  cần phải tính toán trước. Vì vậy, dãy các xử lý là vẫn hiện diện nhưng phải được suy ra bởi hệ thống thì đúng hơn là đã được định nghĩa bởi người lập trình. Dĩ nhiên, sự chăm sóc đặc biệt phải được nắm giữ bởi hệ thống mà thông báo độc lập là đã được nhận diện và được thông báo như là các sự cố (errors).

### **2. Sự trừu tượng hoá thủ tục (Procedural Abstraction)**

Chúng ta phân biệt hai mức độ của việc trừu tượng hoá thủ tục. Những ngôn ngữ lập trình trực quan bậc cao, nghĩa là không thể viết và bảo quản toàn bộ chương trình trong một ngôn ngữ và chắc chắn có một số lớp chứa những mô-đun không trực quan mà chúng được kết hợp lại bằng cách sử dụng một ngôn ngữ trực quan.

Cách tiếp cận này để lập trình trực quan là có thể được tìm thấy trong rất nhiều các hệ thống phục vụ cho một lĩnh vực đặc biệt, ví dụ như những công cụ bảo quản phần mềm và môi trường trực quan khoa học. Ở phía ngược lại là những ngôn ngữ trực quan mức độ thấp, chúng không cho phép người lập trình móc nối những đơn vị lô-gíc đến các mô-đun thủ tục.

Phương pháp này cũng có lợi trong nhiều ngôn ngữ thuộc một lĩnh vực đặc biệt ví dụ như là những bộ mô phỏng lô-gíc. Những ngôn ngữ lập trình trực quan đa năng thường bao phủ một phổ

rộng và cung cấp tất cả những chức năng nhằm giúp người lập trình dễ dàng khi phát triển ứng dụng như móc nối đến các ngôn ngữ bậc thấp, các điều kiện, đệ qui, lặp và móc nối dễ dàng đến các mô-đun trừu tượng (thủ tục, lớp, thư viện...).

### **3. Sự trừu tượng hoá dữ liệu (Data Abstraction)**

Những phương tiện cho phép trừu tượng hoá dữ liệu là chỉ có thể tìm thấy trong các ngôn ngữ lập trình trực quan đa năng. Khái niệm trừu tượng hoá dữ liệu trong lập trình trực quan là rất giống với khái niệm trừu tượng hoá dữ liệu trong những ngôn ngữ lập trình thông thường với chỉ những yêu cầu trừu tượng các kiểu dữ liệu đã được định nghĩa trực quan (trái với textually), có một sự trình bày trực quan (iconic) và được cung cấp để dành cho cách đối xử tương tác.

## **VI. Các ngôn ngữ lập trình trực quan**

Trong phần này, chúng tôi giới thiệu một số ngôn ngữ lập trình trực quan mà chúng là những minh hoạ xác thực cho những khái niệm đã được giới thiệu ở trên.

### **1. ARK**

Hơn 10 năm kể từ ngày bắt đầu, ngôn ngữ Alternate Reality Kit (ARK) được thiết kế bởi R. Smith tại Xerox PARC, đã ra đời một ngôn ngữ lập trình thuộc họ VPL để phục vụ cho một lĩnh vực đặc biệt. ARK được phát triển trên Smalltalk-80, cung cấp cho người dùng một môi trường động trong không gian 2 chiều dành để khởi tạo sự mô phỏng tương tác. Hệ thống là được dự định để phục vụ cho những người lập trình nghiệp dư tạo ra các mô phỏng và để một nhóm khán giả rộng lớn tương tác với những mô phỏng này. Để giúp người sử dụng hiểu những luật cơ bản của tự nhiên, ARK sử dụng phép ẩn dụ theo nghĩa của chữ mà trong đó người sử dụng điều khiển bàn tay trên màn hình mà nó có thể tương tác với các đối tượng vật lý, ví dụ các quả bóng hoặc các khối, ở đó chúng xử lý một khối lượng lớn và vận tốc nhanh và với các đối tượng được gọi là các bộ tương tác, trình bày các qui luật vật lý như là trọng lực. Bằng việc cung cấp một kiểu nguyên bản vật lý đến các luật trừu tượng, hệ thống cố gắng huỷ bỏ một vài bí mật bao quanh theo những cách mà các luật tương tác với các đối tượng và những cái khác. Những người sử dụng có thể hiệu chỉnh bất kỳ đối tượng nào trong việc sử dụng môi trường, các hộp thông điệp và các nút, nhìn thấy kết quả của những thay đổi của chúng trong thời gian thực. Việc mô phỏng thực hiện trong một “alternate reality” chứa trong một cửa sổ và tất cả được bao quanh một thế giới “siêu thực”. Cấu trúc rất giống một hệ thống các màn hình và cửa sổ theo kiểu giao tiếp người dùng đồ hoạ hiện đại. Người lập trình có thể di chuyển bàn tay giữa nguyên bản luân phiên và những đối tượng tác động ra ngoài sự mô phỏng và đến sự siêu thực bất kỳ lúc nào.

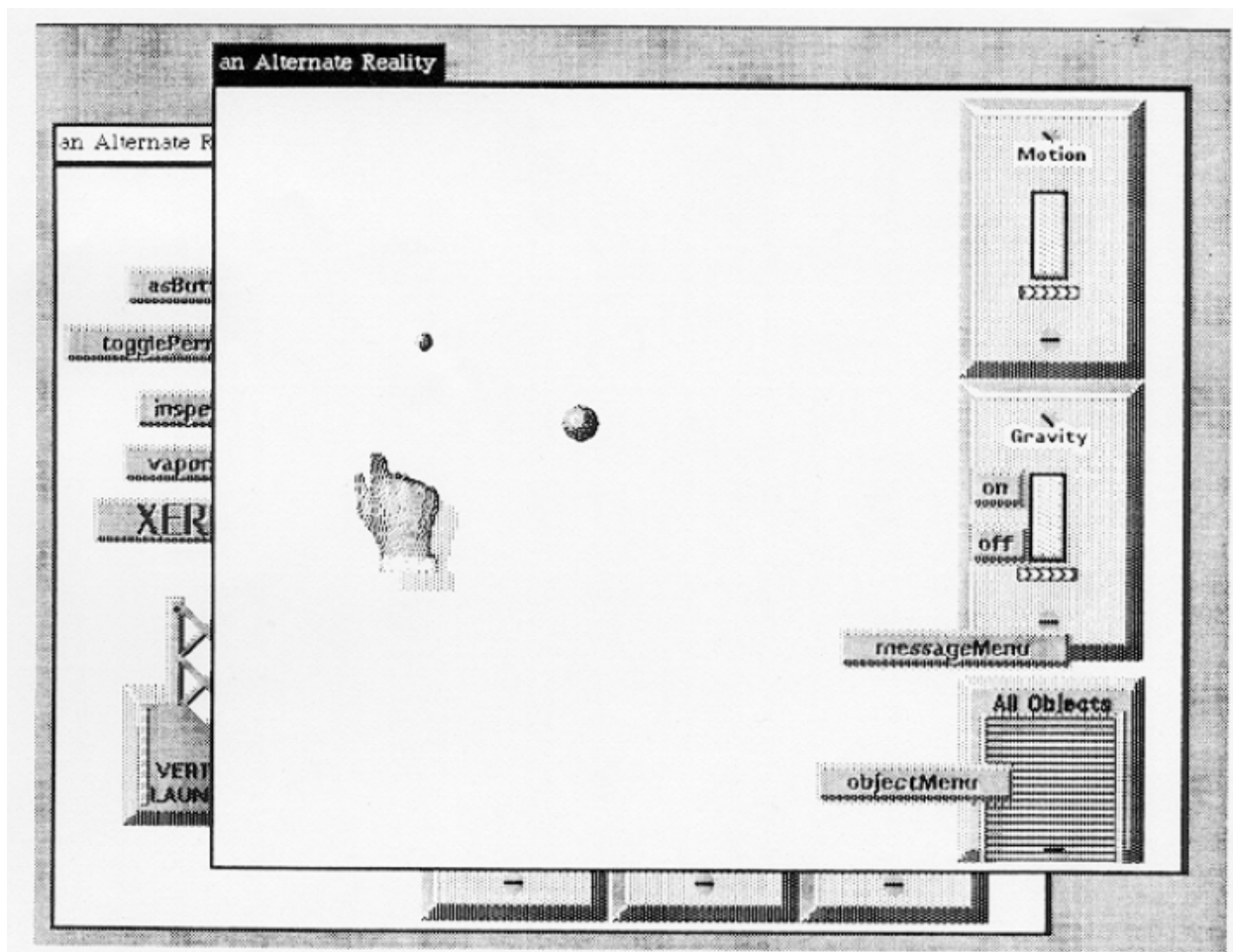
Ví dụ của một người sử dụng vươn tới và sự chuyển động của một đối tượng từ một dãy các hình ảnh xen kẽ là điểm ấn tượng nhất của một trong nhiều điều cần quan tâm khi thiết kế trong ARK Cụ thể là sự cần thiết để đôi khi dừng lại với sự trừu tượng hoá các từ trong trật tự để cung cấp các chức năng có ích. Smith tham chiếu đến vấn đề này như sự quan hệ giữa ảo thuật và sự dịch từng chữ trong ARK. Trong khi sử dụng một bàn tay mà nó có thể “chộp” được các đối tượng vật lý là một điểm sáng của hệ thống, nó cho phép người sử dụng vươn tới một sự mô phỏng và luân phiên hoặc xoá bỏ các đối tượng mà không quan tâm đến các qui luật vật lý hiện hành ở công việc trong môi trường rõ ràng cung ứng cho người sử dụng với cái gì là được xem xét như là năng lực “ảo thuật”. Câu hỏi là khi mà một sự kiện “ảo thuật” hoặc hành động trong ARK tranh chấp với sự trừu tượng hoá vật lý đặt song song với một mô phỏng liên quan trong thiết kế của nhiều ngôn ngữ lập trình trực quan truyền thống.

Khi phát triển những ngôn ngữ lập trình trực quan nhất, các nhà nghiên cứu đã quyết định sử dụng một cách thích hợp các văn bản (text) trong hệ thống của họ.

Trong khi có thể thiết kế một hệ thống không sử dụng bất cứ thứ gì liên quan đến chế độ văn bản và những hệ thống như vậy đã được tạo ra nhưng kết quả là thường khó để đọc và hiểu. Những ngôn ngữ lập trình trực quan nhất thường bao gồm đầy đủ sự trực quan nhưng cũng sử dụng chế độ văn bản cho những vấn đề tối thiểu như tên biến và tên hàm trong các chương trình. Những người thiết kế phải đối mặt với cùng vấn đề như đã gặp phải trong ARK. Họ phải cố gắng cân bằng giữa sự trình bày trực quan và tính dễ sử dụng.

Mặc dù ARK nhắm đến lĩnh vực ứng dụng khá đặc biệt nhưng nó đã hỗ trợ mạnh mẽ cho mô hình lập trình.

Người lập trình có thể không chỉ khởi tạo sự mô phỏng bằng việc liên kết nhiều đối tượng và nhiều tương tác nhưng họ cũng có thể phát triển những bộ tương tác mới. Việc lập trình một mô phỏng, ví dụ như mô phỏng hoạt động của các hành tinh trong hình sau:



Hình 2. Giao diện của hệ thống ARK

Trong hình này bao gồm trước hết việc phát sinh các đối tượng vật lý, giống như quả bóng, từ kho chứa đối tượng trong góc phải bên dưới của màn hình trên. Bằng việc nhấp chuột lên trên nút **objectMenu**, người lập trình có thể chọn để thuyết minh cho một đối tượng bất kỳ trong môi trường. Sau khi khởi tạo một vài đối tượng vật lý, người lập trình đi theo cùng một thủ tục để đặt các tương tác, như sự chuyển động, lực hút giữa các đối tượng trong các mối quan hệ qua lại. Bây giờ người lập trình sử dụng nút **messageMenu** để tìm ra cách làm thế nào

để sắp xếp các thông điệp để các bộ tương tác trả lời bằng cách định vị các nút trên bộ tương tác và nhấn nó với bàn tay.

Cái này sinh ra một danh sách tất cả các thông điệp (message) phù hợp đến bộ tương tác của nó. Người lập trình chọn một cái, ví dụ chọn “off” dành cho bộ tương tác trọng lực (lực hút) trong mô phỏng hoạt động, và hệ thống sinh ra một hộp thông điệp (message box). Các hộp thông điệp là các đối tượng mà nó có thể gửi và nhận các thông điệp Smalltalk. Người lập trình liên kết hộp thông điệp mới, trong trường hợp của chúng ta một trong số đó sinh ra thông điệp “off”, đến bộ tương tác thích hợp bằng cách gắn liền chúng bằng một đường nét chấm (dotted line). Hộp thông điệp có thể thu nhỏ đến một nút đơn như hình trên. Các bộ tương tác ảnh hưởng đến tất cả các đối tượng trong cùng xác thực luân phiên. Như vậy sau sự đặc tả tất cả các điều khiển cần thiết, người lập trình có thể bắt đầu sự mô phỏng.

Rõ ràng, các bộ tương tác của ARK có cách đối xử rất giống những ràng buộc trên các đối tượng vật lý trong sự xác thực luân phiên.

Vì vậy, việc khởi tạo và hiệu chỉnh các bộ tương tác giống như là các đặc điểm hướng ràng buộc của ARK. Một người lập trình có thể phát sinh các bộ tương tác mới bằng cách tạo ra các mạng lưới các hộp thông điệp. Hãy xem một ví dụ đơn giản, xem xét việc phát triển một bộ tương tác lực ma sát bằng cách tạo ra một hộp thông điệp ở đó bổ sung một lực đến một đối tượng ma sát là ngược với vận tốc của nó. Hộp thông điệp có thể thiết lập việc gửi liên tục các thông điệp của nó và khi cách đối xử của nó đã được xác nhận, người lập trình có thể biến đổi nó thành một bộ tương tác.

## **2. Prograph**

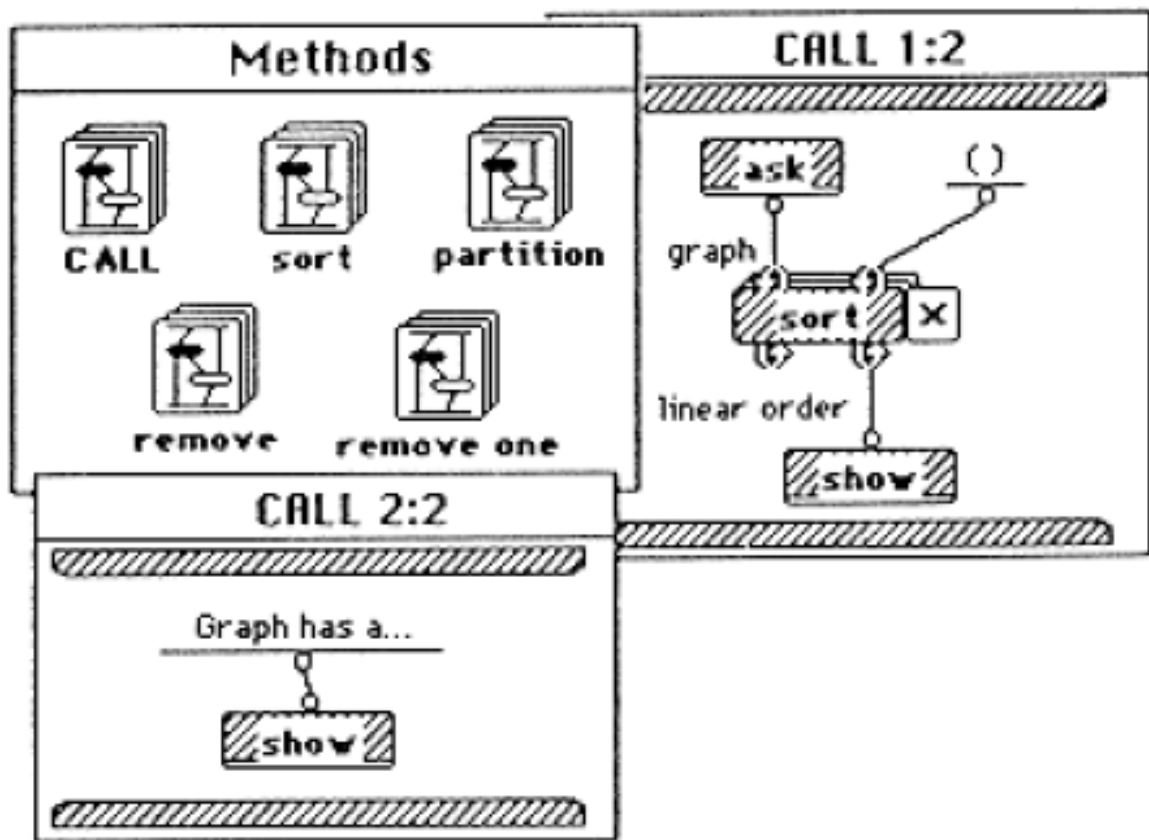
Trong phần này, chúng tôi mô tả ngôn ngữ Prograph, nó được xem là ngôn ngữ thành công nhất trong số các ngôn ngữ trực quan đa năng (general-purpose visual languages).

Việc nghiên cứu Prograph được tiến hành từ năm 1982 tại Trường Đại học Kỹ thuật Nova Scotia. Từ đó đến nay, có vài phiên bản đã được đưa vào sử dụng và trong số đó được dùng phổ biến nhất là Prograph/CPX và đã được thương mại hoá bởi Pictorius, Inc.

Prograph là ngôn ngữ hướng đối tượng trực quan. Nó móc nối những khái niệm về các lớp, các đối tượng với một cơ chế đặc tả dòng dữ liệu trực quan rất mạnh. Prograph còn là một ngôn ngữ mệnh lệnh, nó cung cấp những điều khiển rõ ràng trên trật tự các tính toán. Điều đặc biệt là các trường hợp đơn lẻ và đa thành phần của Prograph, những cấu trúc điều khiển đặc biệt ở đó là được mong đợi để thay thế vòng lặp và cung cấp các điều khiển luồng tinh vi. Chúng ta sẽ thảo luận những điều này như một phần của ví dụ dưới đây.

Prograph cho phép người lập trình làm việc trên cả hai mức độ thấp và cao, cho phép họ thiết kế và bảo trì nhiều hoặc ít phần mềm phức tạp. Những sự tính toán nguyên thủy như các phép toán số học, các lời gọi các method... là đã được móc nối đến các phần thân của phương pháp biểu mẫu bởi các biểu đồ dòng dữ liệu. Các phương thức (methods) là được tổ chức thành các lớp. Ngoài ra, Prograph còn cung cấp cho người lập trình cơ chế gọi những đối tượng mà nó có thể là được lưu trữ trong một cơ sở dữ liệu Prograph giữa các lời đề nghị khác nhau của chương trình.

Chúng ta sẽ giới thiệu ở đây một số đặc điểm thú vị của Prograph thông qua một ví dụ của giải thuật sắp xếp hình học tô-pô trên đồ thị có hướng. Những đồ thị sẽ được trình bày bởi các danh sách gần kề, mỗi một giá trị tương ứng đến một nút. Một danh sách trình bày một nút dựa trên tên của nút, dựa theo các tên của tất cả các nút bắt đầu từ đỉnh của đỉnh ngoài cùng của nút này.



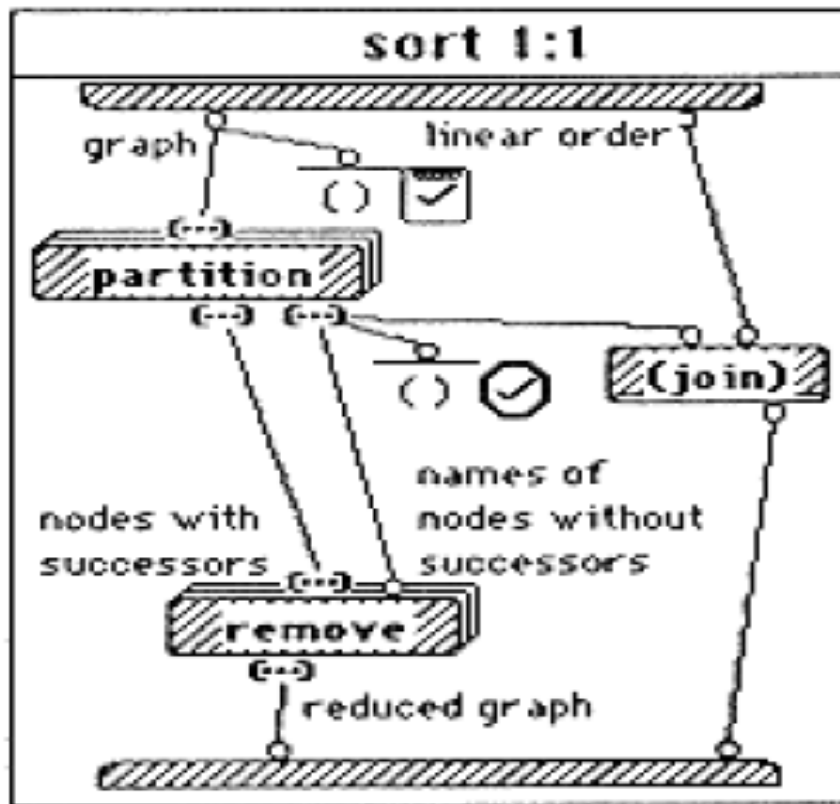
Hình 3. Ví dụ minh họa về giải thuật sắp xếp hình học tô-pô của Prograph

Hình trên mô tả một chương trình Prograph để thực hiện sắp xếp hình học tô-pô. Cửa sổ “Methods” chứa tên của tất cả các chức năng trong chương trình này và với cửa sổ “CALL” đang thể hiện các lộ trình xử lý theo giải thuật.

Cửa sổ “Methods” chứa một hoặc nhiều trường hợp xử lý. Mỗi trường hợp là một biểu đồ dòng dữ liệu nó chỉ rõ làm thế nào các trường hợp xử lý (case) đã được thực thi. Mỗi một mệnh lệnh dòng dữ liệu được chỉ định rõ một sự xử lý để thực hiện trên dữ liệu mà dữ liệu này nhập vào thông qua một hoặc nhiều terminals (thể hiện trên đỉnh của biểu tượng hoạt động). Đầu ra của một sự hoạt động (operation) là kéo dài đến một hoặc nhiều gốc (ở đáy của biểu tượng hoạt động). Các đỉnh của một biểu đồ dòng chỉ rõ làm thế nào dữ liệu lan truyền từ gốc của mỗi một *operation* đến *terminals* của cái kế tiếp, nghĩa là trật tự thực hiện là hướng dữ liệu.

Phương thức “CALL” dựa trên hai trường hợp xử lý (2 cases). Số lượng các trường hợp, hay nói cách khác là tổng số các trường hợp, là đã được xác định trong thanh tiêu đề của cửa sổ chứa biểu đồ dòng dữ liệu của phương thức.

Nó chứa những lời gọi đến hai phương thức hệ thống là “ask” và “show” và một lời gọi đến phương thức do người dùng định nghĩa là “sort” như trong hình sau:



Hình 4. Phương thức “sort” cho giải thuật sắp xếp trong Prograph

Sự thực hiện của “CALL” bắt đầu bởi việc đánh giá kết quả của phương thức “ask” và hằng số “()” – danh sách rỗng. “ask” chứa đầu vào từ người sử dụng và nó trở thành giá trị của gốc của nó. Sau sự đánh giá “ask” và hằng số được hoàn tất, kết quả của chúng là được chuyển cho phương thức “sort”, đây là bước xử lý kế tiếp được thực hiện. Lưu ý rằng việc gọi “sort” là đi cùng với một điều khiển được gọi là *next-on-failure* được trình bày bởi một biểu tượng bên trái của biểu tượng *operation*. Điều này ngụ ý rằng nếu việc gọi đến “sort” bị lỗi (nó dẫn đến đồ thị chứa một vòng tròn), việc thực thi của trường hợp 1 của “CALL” có thể bị ngắt và trường hợp 2 sẽ được thực thi.

Trong trường hợp 1 của “CALL”, sự hoạt động của *sort* là bao gồm nhiều phần (nó là được phân biệt bằng hình ảnh một sự hoạt động đơn lẻ bởi một hình vẽ trong hình dạng 3 chiều để làm nổi bật là nó đã thực thi nhiều lần). Tối thiểu một trong các *roots* hoặc *terminals* của các thành phần trên là đã được chú thích với một biểu tượng mà nó sẽ biểu thị trong kiểu Forms/3.

### 3. Form/3

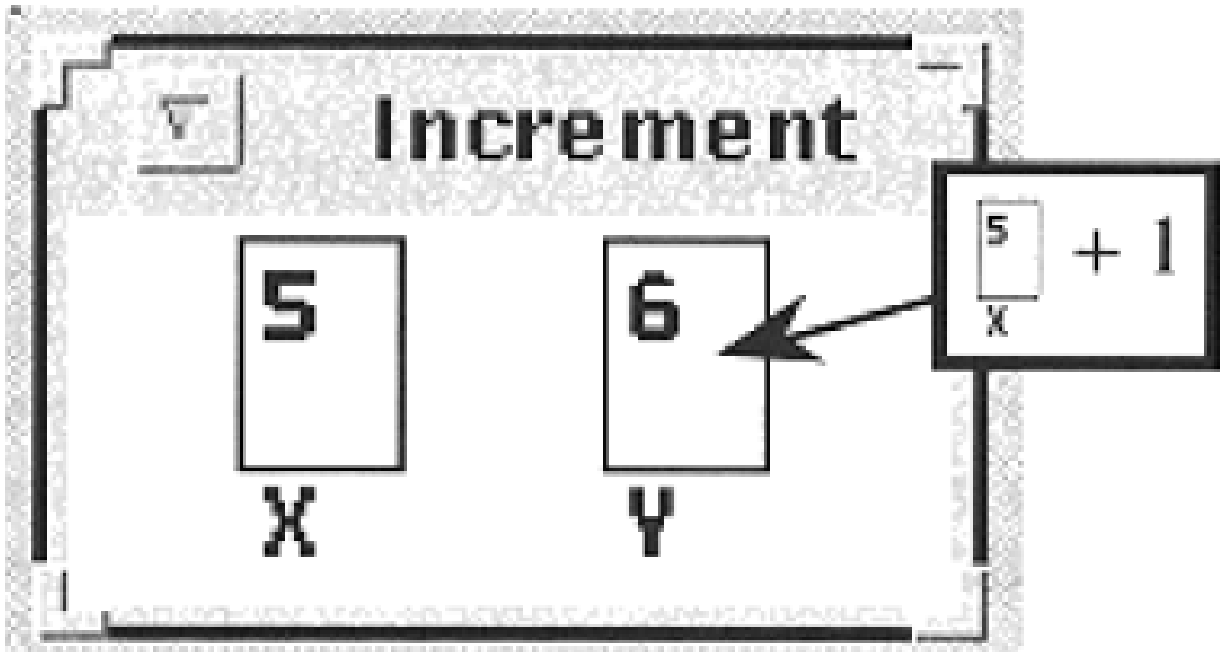
Forms/3 là một ngôn ngữ lập trình khác thuộc dạng ngôn ngữ lập trực quan hướng đối tượng đa năng. Đặc điểm nổi bật của nó là trừu tượng hoá dữ liệu. Tuy nhiên, không giống như Prograph, nó không mang tính kế thừa và sự phân tích các thông điệp là có được hỗ trợ.

Forms/3 phỏng theo cách tổ chức bảng tính với các ô và công thức để trình bày dữ liệu và thứ tự tính toán. Một đặc trưng của Forms/3 là các ô ở đây đã được tổ chức thành một nhóm gọi được là *form*, một cơ chế trừu tượng hoá dữ liệu cơ bản. Một *form* có thể trình bày một ảnh cho trước (còn gọi là một biểu tượng) và nó có thể được minh hoạ cho một đối tượng. Theo nghĩa này, một *form* tương ứng đến một đối tượng nguyên mẫu trong các ngôn ngữ hướng đối tượng nguyên mẫu cơ bản (prototype-based object oriented languages).



Trong Forms/3, dữ liệu (values) và sự tính toán (formulas) là một cặp thống nhất. Mỗi một đối tượng được đặt trong một ô và đã được định nghĩa với khai báo để sử dụng một công thức. Các đối tượng có thể chỉ được khởi tạo theo các công thức và mỗi một công thức sản sinh một đối tượng giống như một kết quả của việc đánh giá nó. Những công thức cung cấp một sự dễ dàng để yêu cầu các kết quả từ các đối tượng khác và khởi tạo những đối tượng mới: điều này không cần sự phân tích các thông điệp.

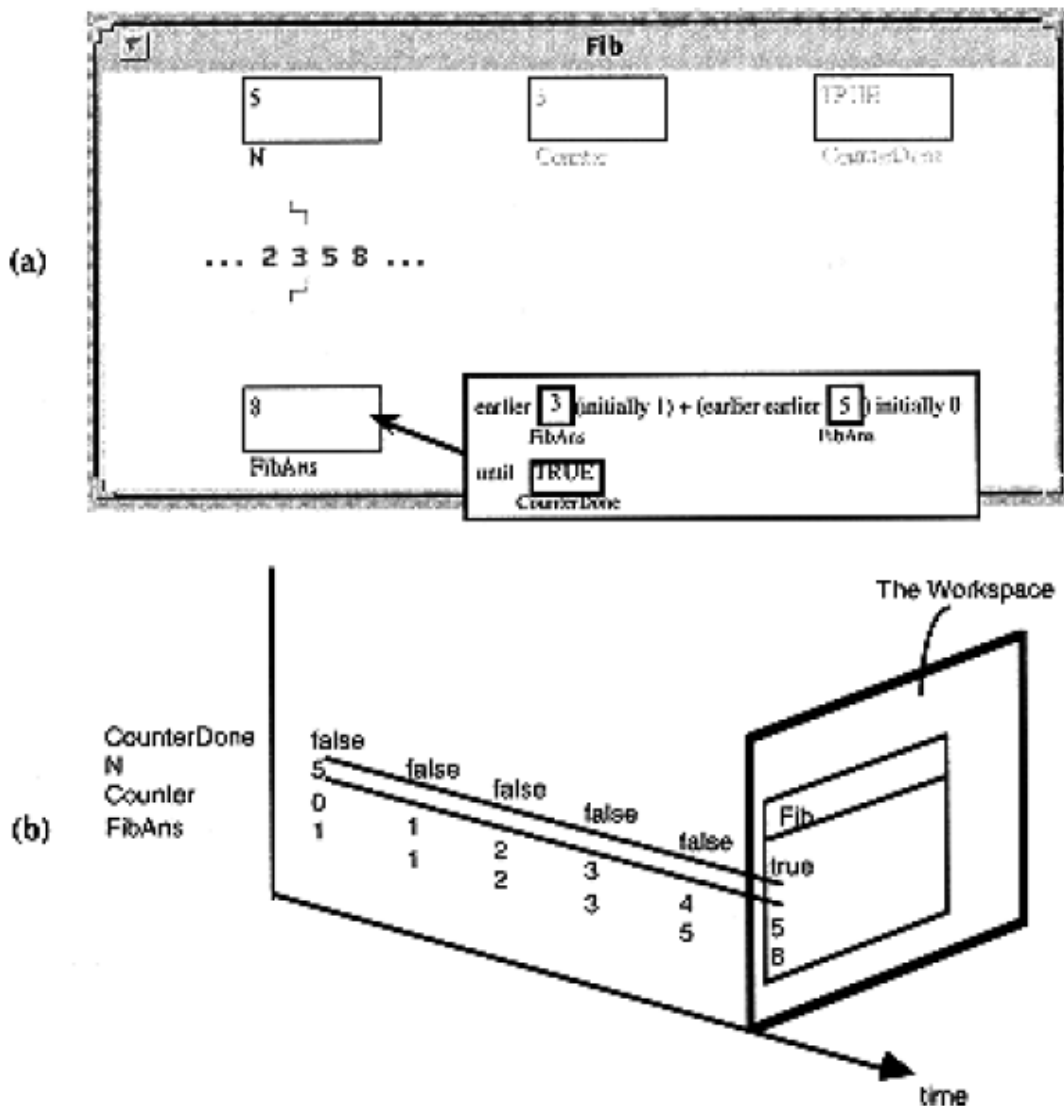
Người lập trình có thể khởi tạo một chương trình mới trong Forms/3 bằng cách khởi tạo một *form* mới, bổ sung các ô đến nó và ghi rõ các công thức. Một *form* đơn giản là được mô tả trong hình sau:



Hình 5. Một form đơn giản trong Forms/3

Các công thức dành cho *form* này đã được đặc tả bởi việc chọn ô “X”, gõ vào số “5”, chọn tiếp ô “Y”, nhấp chuột trên “X” rồi gõ “+ 1”. Người lập trình cũng có thể tham chiếu đến ô “X” bằng việc gõ tên của chúng, đúng hơn là chọn lựa nó trên màn hình.

Forms/3 triển khai một cách tiếp cận khai báo đến luồng điều khiển đã được móc nối với chiều thời gian trong một cách tiếp cận mà các tác giả gọi là véc-tơ thời gian (vectors in time). Với cách tiếp cận này, mỗi một véc-tơ định nghĩa một dãy các đối tượng mà chúng trình bày giá trị của các ô này tại những thời điểm khác nhau theo thời gian. Quay trở lại với form ví dụ trong hình trên. Nếu X định nghĩa một véc-tơ thời gian của các đối tượng số như là  $\langle 1 \ 2 \ 3 \ 4 \ 5 \rangle$ , thì Y định nghĩa một véc-tơ thời gian  $\langle 2 \ 3 \ 4 \ 5 \ 6 \rangle$ . Forms/3 cung cấp bộ lập trình với sự truy cập đến chiều thời gian và như vậy việc lập có thể là được thi hành rất dễ dàng với phương pháp khai báo này. Xem xét ví dụ trong hình sau:



Hình 6. Form tính toán các số Fibonacci trong Forms/3

Giống như những ngôn ngữ lập trình trực quan đa năng (general-purpose visual programming languages), Forms/3 cho phép người lập trình làm việc trên cả hai bậc thấp và cao. Lập trình bậc thấp trong Forms/3 là thực hiện theo các công thức (formulas), trong khi sự trừu tượng bậc cao là được thực hiện bởi sự sưu tập các ô vào trong các *form*.

## VII. Kết luận

Trong lĩnh vực của các ngôn ngữ lập trình trực quan có rất nhiều nghiên cứu và ví dụ điển hình và chúng có cùng mục đích là nỗ lực để mở rộng ảnh hưởng và làm tăng năng lực của lập trình máy tính. Thông qua nhiều dự án đã được trình bày ở trên, chúng ta nhận thấy các nhà nghiên cứu đã hướng đến một mục đích chung là hoàn thiện tiến trình lập trình. Hơn nữa những nghiên cứu hiện tại tập trung vào cơ sở lý thuyết của lập trình trực quan và những nỗ lực nghiêm túc để phát triển các tiêu chuẩn phân loại hình thức dành cho các ngôn ngữ VPL. Chắc chắn là phạm vi nghiên cứu trong lĩnh vực này đã được mở rộng và lớn mạnh trong 20 năm qua. Những kết quả quan trọng từ các nghiên cứu lý thuyết và những sản phẩm như Sketchpad và Pygmalion vẫn duy trì ảnh hưởng của nó đến nhiều ngôn ngữ VPL hiện đại.

Cùng với sự phát triển của các thiết bị phần cứng hỗ trợ mạnh mẽ cho đồ hoạ máy tính, những bộ xử lý tiếp tục được cải tiến về mặt hiệu quả xử lý và giá thành giảm, những ngôn ngữ lập trình trực quan trong không gian 3 chiều, ví dụ như CUBE, sẽ được chờ đợi từ cộng đồng nghiên cứu. Những ngôn ngữ VPL trong không gian 3 chiều không chỉ nhắm đến các vấn đề thể hiện một khối lượng lớn thông tin trên một màn hình nhỏ mà còn mô tả sự phối hợp giữa các ngôn ngữ lập trình, đồ hoạ máy tính và tương tác người máy mà ở đó nó đã được đạt được những thành tựu đáng khích lệ với một sự khởi đầu tốt đẹp.

## CHƯƠNG 2. LẬP TRÌNH TRỰC QUAN VỚI MS ACCESS

### I. Giới thiệu

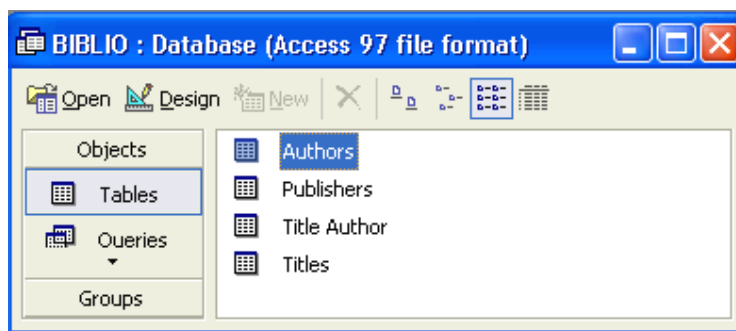
#### 1. Khái niệm về cơ sở dữ liệu

##### a. Table (bảng), Record (bản ghi) và Field (trường)

Nói đến cơ sở dữ liệu, ta nghĩ ngay đến các hệ quản trị cơ sở dữ liệu như Foxpro, DB/2, SQLServer, Access hay Oracle... Cơ sở dữ liệu cho phép lưu trữ dữ liệu với số lượng lớn để ta có thể quản lý, khai thác một cách tiện lợi và nhanh chóng. Hầu hết các chương trình ta viết đều có truy cập cơ sở dữ liệu và ta dùng nó như một công cụ để làm việc dễ dàng với những dữ liệu lớn và có thể tập trung thời gian vào việc lập trình phần giao diện với người dùng. Ta cần có một kiến thức cơ bản về kiến trúc của cơ sở dữ liệu để hiểu lý do tại sao ta thiết kế hay truy cập nó theo những cách nhất định.

Để minh họa, ta sẽ dùng một cơ sở dữ liệu có sẵn của Access là **biblio.mdb** nằm ở **\Program Files\Microsoft Visual Studio\VB98\biblio.mdb** để minh họa các khái niệm cần biết về cơ sở dữ liệu.

Trong cơ sở dữ liệu này có 4 bảng: **Authors** (tác giả), **Publishers** (nhà xuất bản), **Titles** (tên sách) và **Title Author**.



Bảng Authors chứa nhiều bản ghi. Mỗi bản ghi trong bảng Authors chứa 3 trường: **Au\_ID**, **Author** và **Year Born**. Ta có thể trình bày bảng Authors dưới dạng một bảng tính như sau:

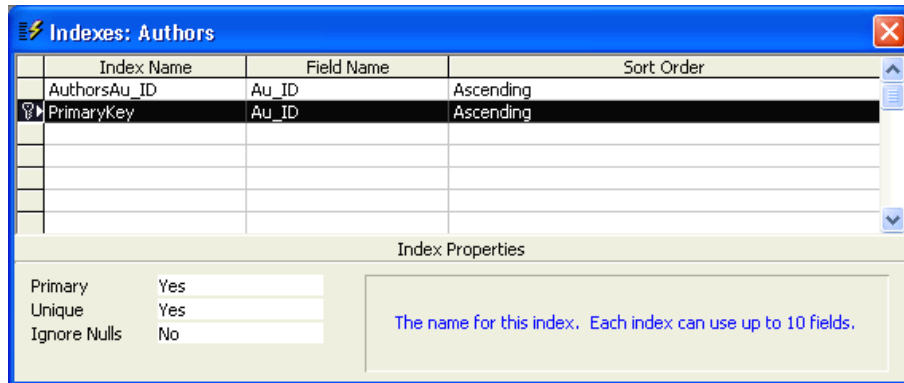
Authors : Table			
	Au_ID	Author	Year Born
+	1	Jacobs, Russell	1950
+	2	Metzger, Philip W.	1962
+	3	Boddie, John	1954
+	4	Sydow, Dan Parks	1963
+	6	Lloyd, John	1948
+	8	Thiel, James R.	1955
+	10	Ingham, Kenneth	1945
+	12	Wellin, Paul	1939
+	13	Kamin, Sam	1947
+	14	Gaylord, Richard	1963
+	15	Curry, Dave	1958
+	17	Gardner, Juanita Mercado	1957

Record: 1 of 6246

Vì nội dung của một trường của các bản ghi hiển thị trong cùng một cột của bảng tính, nên ta cũng nói đến một trường như một cột (column). Và vì mỗi bản ghi chiếm một dòng (row) của bảng tính, nên có khi ta cũng nói đến một bản ghi như một dòng.

### b. Primary Key và Index

Để tránh sự trùng hợp, thường thường có một trường của bản ghi, ví dụ như Au\_ID trong bảng Authors, được dành ra để chứa một trị số duy nhất (unique). Tức là trong bảng Authors chỉ có một bản ghi với trường Au\_ID có trị số ấy mà thôi. Ta gọi nó là Primary Key (khóa chính).

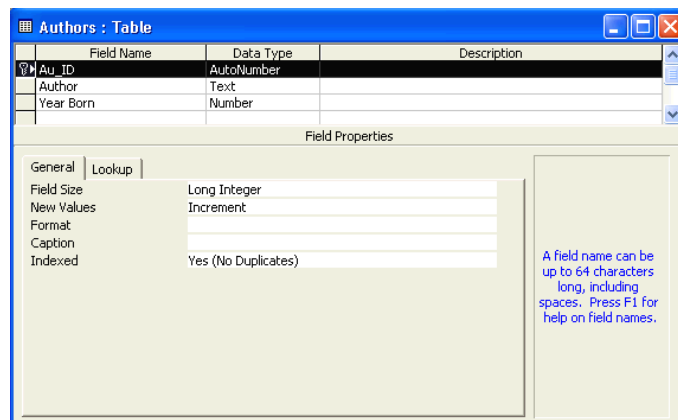


Không phải lúc nào ta cũng muốn truy cập một bản ghi Author dựa vào Au\_ID. Nhiều khi ta muốn dùng chính tên của Author để truy cập, do đó ta cũng cần phải sắp xếp sẵn các bản ghi theo thứ tự alphabet. Ta cũng có thể hợp nhiều trường lại để sắp xếp các bản ghi. Thật ra, chính các bản ghi không cần phải được dời đi (di chuyển vật lý) để nằm đúng vị trí thứ tự. Ta chỉ cần biết vị trí của nó ở đâu trong bảng là đủ.

Trường hay tập hợp của nhiều trường (ví dụ surname và firstname) để dùng vào việc sắp xếp này được gọi là Index (chỉ mục). Một bảng có thể có một hay nhiều Index. Mỗi Index sẽ là một bảng nhỏ của những **pointers** (con trỏ), chứa vị trí của các bản ghi trong bảng Authors. Nó giống như mục lục của một cuốn sách chứa đề mục và số trang tương ứng để chỉ ta đến đúng phần ta muốn tìm trong quyển sách.

Khi thiết kế một bảng ta chỉ định **Datatype** (kiểu dữ liệu) của mỗi trường để có thể kiểm tra dữ liệu nhập vào có hợp lệ hay không. Các kiểu dữ liệu thông dụng là Number, String (để chứa Text), Boolean (Yes/No), Currency (để chứa trị số tiền) và Date (để chứa date/time). Kiểu dữ liệu Number lại gồm có nhiều loại kiểu dữ liệu về số như Integer, Long (integer chiếm 32 bits), Single, Double...

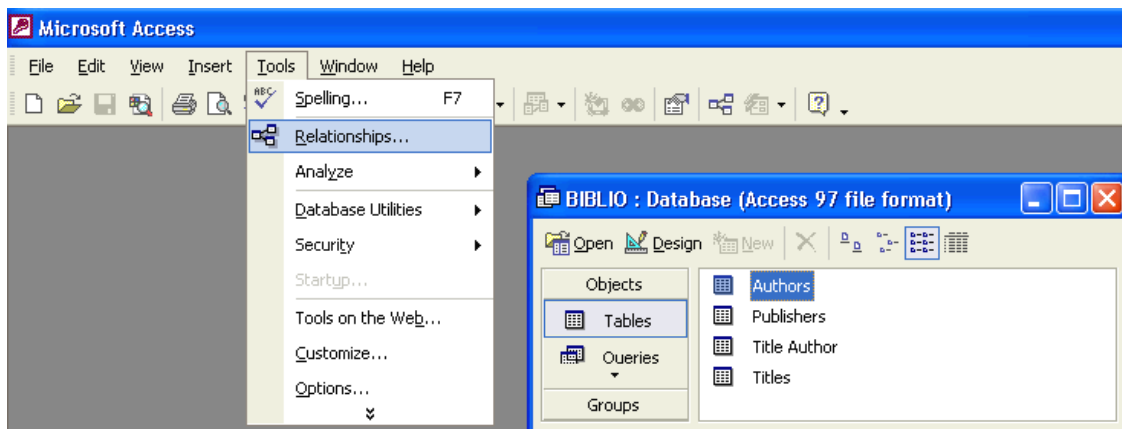
Dưới đây là các kiểu dữ liệu của các trường trong bản ghi Author:



Có loại kiểu dữ liệu đặc biệt tên là **AutoNumber**. Thật ra nó là kiểu Long nhưng trị số được phát sinh tự động mỗi khi ta thêm một bản ghi mới vào bảng. Ta không làm gì hơn là phải chấp nhận con số ấy.

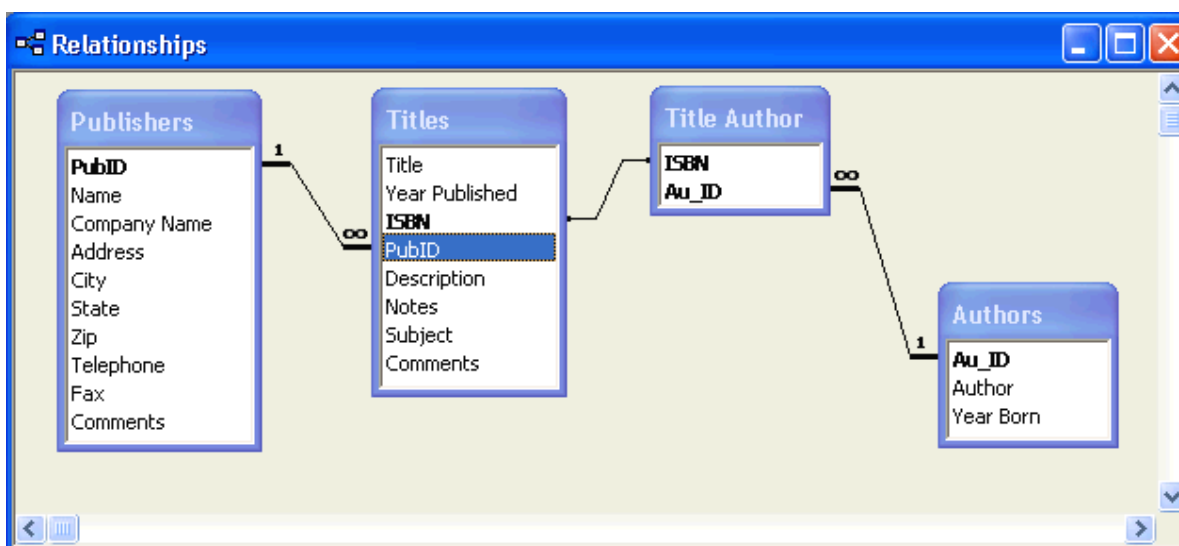
**c. Relationship (quan hệ) và Foreign Key (khoá ngoại)**

Bây giờ, nếu chúng ta đang chạy Microsoft Access để quan sát cơ sở dữ liệu biblio.mdb, chúng ta có thể dùng Menu Command **Tools | Relationships** như sau để xem sự liên hệ (relationships) giữa các bảng.



Access sẽ hiển thị hộp thoại Relationships, trong đó mỗi bảng có chứa tên các trường. Mỗi bảng lại có một hay hai sợi dây nối qua các bảng khác. Mỗi sợi dây là một mối liên hệ (relationship), nó nối một trường trong một bảng với một trường có cùng tên trong bảng kia.

Ví dụ như giữa hai bảng **Publishers** và **Titles** có mối liên hệ dựa trên trường **PubID** (**P**ublisher **I**Dentification – mã số của nhà xuất bản). Hơn nữa, nếu để ý chúng ta sẽ thấy ở đầu dây phía bảng Publishers có con số **1**, còn ở đầu dây bên phía bảng Titles có dấu vô cực ( $\infty$ ). Ta gọi mối liên hệ (**1- $\infty$** ) là **one-to-many**, ý nói **một** nhà xuất bản có thể phát hành **nhiều** đề mục sách/CD.



Tương tự như vậy, trong mối liên hệ one-to-many giữa bảng Authors và Title Author, ta thấy một tác giả (bên đầu có con số 1) có thể có nhiều cuốn sách được đại diện bởi các bản ghi thuộc bảng Title Author.

Trong khi đó giữa hai bảng Titles và Title Author, ta có một mối liên hệ one-to-one, tức là tương ứng với mỗi bản ghi thuộc Title chỉ có một bản ghi tương ứng thuộc Title Author. Câu hỏi đặt ra là các mối liên hệ one-to-many đóng vai trò gì trong thiết kế cơ sở dữ liệu.

Tương tự khi ta làm việc với bảng Titles, nhiều khi ta muốn biết chi tiết của nhà xuất bản của cuốn sách ấy. Thật ra ta đã có thể chứa chi tiết của nhà xuất bản của mỗi cuốn sách ngay trong bảng Titles. Tuy nhiên, làm như thế có điểm bất lợi là bản ghi của các cuốn sách có cùng nhà xuất bản sẽ chứa những dữ liệu giống nhau. Mỗi lần muốn sửa đổi chi tiết của một nhà xuất bản ta phải sửa chúng trong mỗi bản ghi của Title thuộc nhà xuất bản ấy. Vì muốn chứa chi tiết của mỗi nhà xuất bản ở một chỗ duy nhất, tránh sự lặp lại, nên ta đã chứa chúng trong một bảng riêng, tức là bảng Publishers.

Nếu giả sử ta bắt đầu thiết kế cơ sở dữ liệu với bảng Titles, rồi quyết định tách các chi tiết về nhà xuất bản để vào một bảng mới, tên Publishers, thì kỹ thuật ấy được gọi là sự chuẩn hoá. Nói một cách khác, chuẩn hoá là thiết kế các bảng trong cơ sở dữ liệu làm sao để mỗi loại mảnh dữ liệu (không phải là Key) chỉ xuất hiện ở một nơi duy nhất.

Trong mối liên hệ one-to-many giữa bảng Publishers và Titles, trường PubID là Primary Key trong bảng Publishers. Trong bảng Titles, trường PubID được gọi là Foreign Key, có nghĩa rằng đây là Primary Key của một bảng lạ (foreign). Hay nói một cách khác, trong khi làm việc với bảng Titles, lúc nào cần chi tiết một nhà xuất bản, ta sẽ lấy khóa lạ (Foreign Key) dùng làm Primary Key của bảng Publishers để truy cập bản ghi ta muốn. Để ý là chính bảng Titles có Primary Key ISBN của nó.

#### d. *Relational Database (cơ sở dữ liệu quan hệ)*

Một cơ sở dữ liệu có nhiều bảng và hỗ trợ các liên hệ, nhất là one-to-many, được gọi là **Relational Database**. Khi thiết kế một cơ sở dữ liệu, ta sẽ tìm cách sắp đặt các dữ liệu từ thế giới thật bên ngoài vào trong các bảng. Ta sẽ quyết định chọn các cột (columns/fields) nào, chọn Primary Key, Index và thiết lập các mối liên hệ, tức là đặt các Foreign Key ở đâu.

Trong số các lợi ích của một thiết kế Relational Database có:

- Sửa đổi dữ liệu, cho vào bản ghi mới hay xoá bỏ (delete) bản ghi có sẵn rất hiệu quả.
- Truy cập dữ liệu, làm báo cáo (Reports) cũng rất hiệu quả. Vì dữ liệu được sắp đặt thứ tự và có quy củ nên ta có thể tin cậy ở cơ sở dữ liệu. Hơn nữa, hầu hết dữ liệu nằm trong cơ sở dữ liệu thay vì trong chương trình ứng dụng, nên cơ sở dữ liệu tự có các tài liệu giải thích kèm theo.
- Dễ sửa đổi chính cấu trúc của các bảng.

#### e. *Integrity Rules (luật toàn vẹn dữ liệu)*

**Integrity Rules** được dùng để nói về những qui luật cần phải tuân theo trong khi làm việc với cơ sở dữ liệu để đảm bảo là cơ sở dữ liệu còn tốt. Có hai loại quy luật: luật toàn vẹn tổng quát (General Integrity Rules) và luật toàn vẹn riêng cho cơ sở dữ liệu (Database-Specific Integrity Rules). Các luật riêng này thường tùy thuộc vào các quy luật về giao dịch (Business Rules).

#### **General Integrity Rules**

Có hai luật toàn vẹn liên quan đến cơ sở dữ liệu: luật toàn vẹn thực thể (Entity Integrity Rule) và luật toàn vẹn tham chiếu (Referential Integrity Rule).

**Entity Integrity Rule** nói rằng **Primary Key** không thể thiếu được, tức là không thể có trị số **NULL**. Quy luật này là xác đáng vì mỗi Primary Key đưa đến một dòng duy nhất trong bảng, nên dĩ nhiên nó phải có một trị số xác định.

Lưu ý là Primary Key có thể là một **Composite Key** (khoá kết hợp), tức là tập hợp của một số khoá (columns/fields), nên nhất định không có khoá nào trong số các cột là **NULL** được.

**Referential Integrity Rule** nói rằng cơ sở dữ liệu không thể chứa một Foreign Key mà không có Primary Key tương ứng của nó trong một bảng khác. Điều ấy hàm ý rằng:

- Ta không thể thêm một dòng vào trong một bảng với trị số Foreign Key trong dòng ấy không tìm thấy trong danh sách Primary Key của bảng bên phía **one** (1) mà nó liên hệ.
- Nếu có thay đổi trị số của Primary Key của một dòng hay xóa một dòng trong bảng bên phía **one** (1) thì ta không thể để các bản ghi trong bảng bên phía **many** ( $\infty$ ) chứa những dòng trở thành mồ côi (orphans).

Nói chung, có ba tùy chọn (options) ta có thể chọn khi thay đổi trị số của Primary Key của một dòng hay xóa một dòng trong bảng bên phía **one** (1):

- **Disallow** (không cho làm): Hoàn toàn không cho phép chuyện này xảy ra.
- **Cascade** (ảnh hưởng dây chuyền): Nếu trị số Primary Key bị thay đổi thì trị số Foreign Key tương ứng trong các bản ghi của bảng bên phía **many** ( $\infty$ ) được thay đổi theo. Nếu dòng chứa Primary Key đã bị xoá thì các bản ghi tương ứng trong bảng bên phía **many** ( $\infty$ ) bị xoá theo.
- **Nullify** (cho thành NULL): Nếu dòng chứa Primary Key bị xoá thì trị số Foreign Key tương ứng trong các bản ghi của bảng bên phía **many** ( $\infty$ ) được đổi thành NULL, để hàm ý đừng có đi tìm thêm chi tiết ở đâu cả.

### Database-Specific Integrity Rules

Những quy luật toàn vẹn nào khác không phải là Entity Integrity Rule hay Referential Integrity Rule thì được gọi là Database-Specific Integrity Rules. Những quy luật này dựa vào chính loại cơ sở dữ liệu và nhất là tùy thuộc vào các quy luật về mậu dịch (Business Rules) ta dùng cho cơ sở dữ liệu, ví dụ như mỗi bản ghi về tiền lương của công nhân phải có một trường Số Thuế (Tax Number) do đơn vị thuế qui định cho người đóng thuế. Lưu ý là các quy luật này cũng quan trọng không kém các quy luật tổng quát về toàn vẹn. Nếu ta không áp dụng các Database-Specific Integrity Rules nghiêm chỉnh thì cơ sở dữ liệu có thể bị hỏng và không còn dùng được.

## 2. Microsoft Access

Microsoft Access là một phần mềm quản lý cơ sở dữ liệu rất mạnh và được sử dụng rộng rãi hiện nay. Nó cho phép người sử dụng lưu trữ, quản lý và khai thác dữ liệu trên máy tính một cách dễ dàng và hiệu quả.

Access nằm trong bộ Microsoft Office của công ty Microsoft. Trong chương trình này chúng tôi giới thiệu trên phiên bản Access 98, đây là phiên bản mới có nhiều cải tiến so với các phiên bản trước đây.

Để sử dụng được Access 98, máy tính phải thỏa mãn các yêu cầu cơ bản sau :

- CPU Pentium trở lên.
- Bộ nhớ RAM 32 MB trở lên.



- Hệ điều hành Windows 95 trở đi.

Trong phiên bản này chúng ta được hưởng một số công cụ bổ sung so với các phiên bản cũ trước đây như : truy cập dữ liệu Access từ các trang Web, quản lý các tập tin có chứa các liên kết đến những tập tin khác, hỗ trợ đa ngữ, quản lý dễ dàng các đối tượng đồ họa, sử dụng các Macro hỗ trợ cho tự động hóa việc quản lý dữ liệu...

### 3. Khởi động ACCESS

Sau khi đã cài đặt Microsoft Office (chọn component Access), mỗi lần làm việc với Access chúng ta khởi động :

- Bật máy tính
- Chọn Start --> Program --> Microsoft Access (*Hoặc nhấn đúp chuột tại biểu tượng của Access trên Desktop*).

### 4. Cơ sở dữ liệu trong Access

Cơ sở dữ liệu là một tập hợp các dữ liệu liên quan đến một chủ đề hay một mục đích quản lý nào đó. Các thành phần của cơ sở dữ liệu Access bao gồm :

- TABLE (bảng) : là thành phần cơ bản của cơ sở dữ liệu, nó cho phép lưu trữ dữ liệu phục vụ công tác quản lý. Trong một bảng, số liệu được tổ chức thành các trường (Field) và các bản ghi (Record).
- QUERY (vấn tin) : là công cụ để truy vấn thông tin và thực hiện các thao tác trên dữ liệu. Query cho phép liên kết các dữ liệu từ nhiều bảng khác nhau, chọn lựa các thông tin cần quan tâm, nó là nền tảng để xây dựng các báo cáo theo yêu cầu thực tế.
- FORM (mẫu) : cho phép xây dựng các mẫu nhập số liệu giống như trong thực tế. Ta có thể cùng lúc nhập số liệu vào nhiều bảng khác nhau thông qua SubForm.
- REPORT (báo cáo) : là các báo cáo số liệu để thông báo kết quả cho người sử dụng. Trong Report ta có thể kết hợp với Query để tạo các báo cáo theo những yêu cầu khác nhau trong thực tế. Trên Report bao gồm số liệu, hình ảnh, đồ thị... để mô tả cho số liệu.
- MACRO (lệnh ngầm) : là một tập hợp các lệnh nhằm tự động thực hiện các thao tác thường gặp. Khi gọi Macro, Access sẽ tự động thực hiện một dãy các lệnh tương ứng, nó được xem là một cụ lập trình đơn giản, cho phép người sử dụng chọn lựa công việc tùy theo tình huống hiện tại.
- MODULE (đơn thể) : một dạng tự động hóa chuyên sâu hơn Macro, đó là những hàm riêng của người sử dụng được viết bằng ngôn ngữ VBA. Ta chỉ nên sử dụng Module trong trường hợp các Macro không đáp ứng được yêu cầu đó.

### 5. Các phép toán

#### a. Các phép toán Logic

- Not : cho kết quả ngược lại
- And : cho kết quả đúng chỉ khi cả hai đều đúng.
- Or : cho kết quả sai chỉ khi cả hai đều sai.
- Xor : cho kết quả đúng khi hai điều kiện có giá trị trái nhau.
- Epv : cho kết quả đúng chỉ khi hai điều kiện có cùng giá trị.

**b. Các phép toán số học**

- ^ : lũy thừa.
- \* : nhân.
- / : chia
- \ : chia lấy phần nguyên.
- Mod : chia lấy phần dư
- + : cộng
- - : trừ

**c. Các phép toán so sánh : >, >=, <, <=, = và <>**

**d. Dấu rỗng :**

- " ... " : rỗng giá trị chuỗi. Ví dụ : "Nguyễn Văn A"
- [ ... ] : rỗng tên biến. Ví dụ : [diem1] + [diem2]
- #mm/dd/yy# : rỗng giá trị ngày. Ví dụ : #01/01/68#

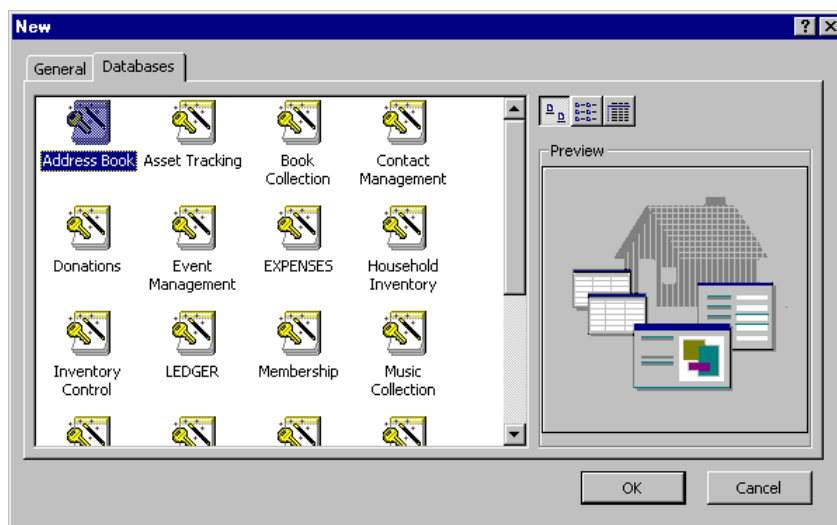
## **II. Làm việc với cơ sở dữ liệu (CSDL)**

### **1. Tạo cơ sở dữ liệu**

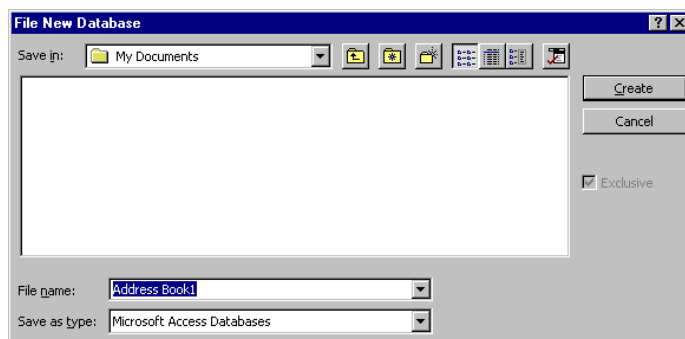
**a. Tạo cơ sở dữ liệu bằng WIZARD**

Cho phép tạo cơ sở dữ liệu theo sự hướng dẫn của ACCESS thông qua các mẫu có sẵn. Thông thường các cơ sở dữ liệu này không phù hợp với cách tổ chức cơ sở dữ liệu thường dùng nên nếu tạo cơ sở dữ liệu theo kiểu này đòi hỏi phải sửa đổi nhiều. Không nên tạo cơ sở dữ liệu theo kiểu này.

- **Bước 1** : ngay sau khi khởi động ACCESS ta chọn vào nút Database Wizard và OK.
- **Bước 2** : lúc đó trên màn hình xuất hiện cửa sổ sau :



- Lúc này ta chọn một mẫu cơ sở dữ liệu ở trên bằng cách Double Click chuột tại biểu tượng tương ứng rồi chọn OK.
- **Bước 3** : lúc đó trên màn hình xuất hiện cửa sổ sau :



Lúc này phải vào tên của cơ sở dữ liệu trong mục : **File name** :, sau đó chọn **Create**

Tiếp tục trên màn hình sẽ xuất hiện các cửa sổ yêu cầu khai báo danh sách các bảng, các Field, kiểu màn hình, các mẫu báo cáo, tiêu đề và biểu tượng của cơ sở dữ liệu...

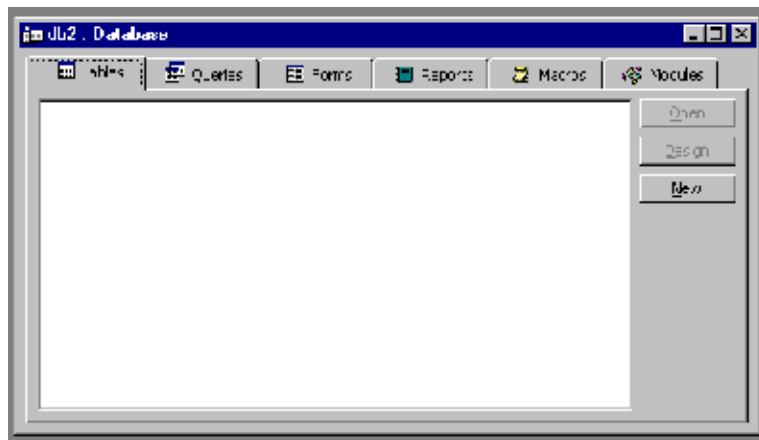
Trong các bước đó ta chỉ việc lựa chọn theo yêu cầu và Double Click vào **Next** để chuyển sang cửa sổ kế tiếp cho đến màn hình cuối thì chọn **Finish**.

#### **b. Tạo cơ sở dữ liệu trống**

Thông thường ta phải sử dụng mục này để tạo một cơ sở dữ liệu cho mình. ACCESS sẽ tạo ra một cơ sở dữ liệu trống và ta tự định nghĩa cho mình các Table, Query, Report, Form, Macro và Module riêng.

- **Bước 1** : khi khởi động chọn Blank Database hoặc chọn File - New Database
- **Bước 2** : khai báo tên của ổ đĩa, thư mục, tập tin cần tạo. Chọn Create

Lúc đó ta nhận được cơ sở dữ liệu mới, xuất hiện màn hình :



Thông thường ta phải sử dụng mục này để tạo một cơ sở dữ liệu cho mình. ACCESS sẽ tạo ra một cơ sở dữ liệu trống và ta tự định nghĩa cho mình các Table, Query, Report, Form, Macro và Module riêng.

- Bước 1 : khi khởi động chọn Blank Database hoặc chọn File - New Database
- Bước 2 : khai báo tên của ổ đĩa, thư mục, tập tin cần tạo. Chọn Create

## **2. Hiệu chỉnh cơ sở dữ liệu**

Sau khi đã tạo cơ sở dữ liệu ta có thể làm việc với cơ sở dữ liệu trên thông qua Table, Report, Form, Record, Query và Module qua cửa sổ trên.

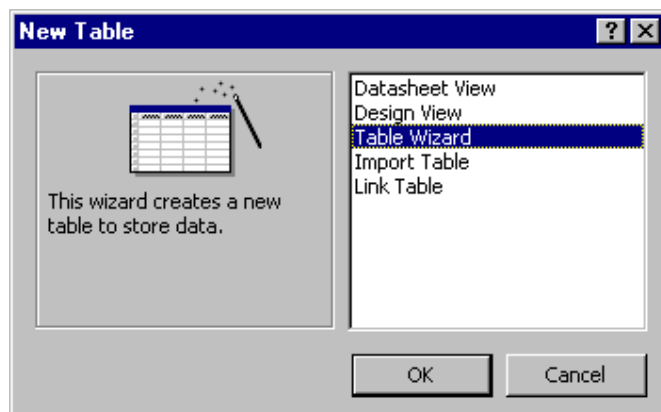
### III. Làm việc với Table

Table là thành phần cơ bản của cơ sở dữ liệu trong Access, nó có nhiệm vụ lưu trữ các số liệu phục vụ quá trình quản lý.

#### 1. Tạo cấu trúc của Table

Để lưu trữ số liệu trên Table trước hết ta phải tạo cấu trúc của Table bằng cách qui định tên của Table, tên và thuộc tính của các trường.

Ta có thể tạo Table bằng cách chọn New trong hộp thoại cơ sở dữ liệu hoặc chọn trên thanh thực đơn Insert - Table, lúc đó xuất hiện cửa sổ cho phép chọn cách tạo Table như sau :



##### a. Tạo Table bằng Wizard

Phương pháp này cho phép tạo Table theo các mẫu có sẵn của Access.

- Bước 1: chọn Table Wizard trong hộp trên rồi OK
- Bước 2: chọn tên Table, tên trường theo mẫu có sẵn của ACCESS và sửa đổi lại theo yêu cầu thực tế. Chọn NEXT để thực hiện các công việc kế tiếp như sửa tên trường, tên Table và sau cùng chọn FINISH để kết thúc.

##### b. Tạo Table bằng DATASHEET VIEW

Phương pháp này cho phép tạo Table theo cách sử dụng một mẫu biểu cho trước và ACCESS dựa vào đó để tạo ra Table.

- Bước 1: chọn Datasheet View trong hộp rồi OK
- Bước 2: Nhập vào nội dung của bảng mẫu khi máy đưa ra một mẫu Table với các Column có tên là Field1, Field2...

Ví dụ : để tạo Table lưu trữ điểm sinh viên ta nhập :

Table1 : Table					
Field1	Field2	Field3	Field4	Field5	Field6
001	Lê Văn Tèo	01/01/72	94T	5	6
002	Bùi Thị Tí	12/01/73	95T	7	8
003	Phan Văn Ti	10/15/76	97T	8	9

Record: 3 of 30

- Bước 3: hiệu chỉnh lại tên trường bằng cách đưa dấu chuột vào đỉnh cột cần sửa và nhấn nút chốt bên phải rồi chọn Rename Column (Hoặc để con trỏ ở ô có cột cần sửa chọn trên thực đơn Format - Reneme Column). Sau đó gõ lại tên trường.

Ví dụ ta nhập lại tên các trường trên Table cũ như sau :

Table1 : Table					
MASO	HOTEN	NGAYSINH	LOP	MON1	MON2
001	Lê Văn Tèo	01/01/72	94T	5	6
002	Bùi Thị Tí	12/01/73	95T	7	8
003	Phan Văn Ti	10/15/76	97T	8	9

Record: 14 3 of 30

Bước 4: đóng Talbe (chọn File - Close)

- Máy hỏi có ghi hay không, chọn Yes. để ghi, No nếu không.
- Đặt tên cho Table trong bảng Save As
- Máy hỏi có đặt khóa cơ sở Primary Key hay không, nếu có thì Yes, không thì No.

Lúc này máy sẽ tự định nghĩa một Table theo mẫu vừa tạo. Nếu muốn hiệu chỉnh thêm thì chọn Design.

Chú ý : tên trường và tên Table dài tối đa là 64 ký tự, bắt đầu bằng 0..9 hoặc A..Z, có thể là ký tự trống nhưng không có dấu chấm câu. Số trường tối đa trong một Table là 255. Độ lớn tối đa một Table là 1 GB.

### c. Tạo Table bằng DESIGN VIEW

Phương pháp này cho phép tạo Table hoàn toàn do người sử dụng qui định.

- Bước 1: chọn Design View trong hộp rồi OK
- Bước 2: xuất hiện màn hình thiết kế Table như sau :

- Field name : khai báo tên của trường.
- Data Type : khai báo kiểu dữ liệu tương ứng của trường.

- Description : nội dung mô tả cho trường. Nội dung được dùng làm tiêu đề cho trường khi thiết lập các Form hay Report khi dùng Wizard.

Trong mục Data Type, chúng ta có thể chọn một trong các kiểu sau :

Tên	Ý nghĩa
Text	Chứa tập hợp các ký tự tùy ý, dài tối đa 255 ký tự
Memo	Dài tối đa 65535 ký tự
Number	Chứa giá trị số
Date/Time	Giá trị ngày hoặc giờ
Currency	Tiền tệ, có đơn vị tính
Auto Number	Giá trị số nhưng không thay đổi được dạng thể hiện
Yes/No	Giá trị True hoặc False
Hyperlink	Nội dung là văn bản hay kết hợp giữa văn bản và số được sử dụng như một địa chỉ hyperlink (siêu liên kết)
Lookup Wizard	Chọn một giá trị trong danh sách các giá trị cho trước

Chú ý : tương ứng với mỗi kiểu dữ liệu sẽ khai báo thêm các thuộc tính của nó trong Field Properties gồm các thuộc tính chung (General) và thuộc tính nhập số liệu (Lookup).

Ví dụ : với kiểu dữ liệu Text ta khai thêm :

Tên	Ý nghĩa
Field Size	Độ rộng tối đa chứa sẵn
Format	Cách hiển thị giá trị
Input Mask	Qui định mẫu nhập liệu
Caption	Một chú thích khác cho Field, dùng với Form, Report
Default Value	Giá trị cho trước
Validation Rule	Qui định cách kiểm tra số liệu nhập
Validation Text	Thông báo khi nhập số liệu sai
Required	Chọn Yes nếu bắt buộc phải nhập nội dung
Allow Zero Length	Chọn Yes nếu chấp nhận giá trị rỗng
Indexed	Có chỉ mục hay không, nếu có thì được trùng hay không (No, Yes Duplicate OK, Yes No Duplicate)

## 2. Nhập số liệu vào Table

Sau khi đã tạo xong Table ta có thể nhập số liệu vào đó bất kỳ lúc nào bằng cách :

- Double Click vào tên Table cần nhập.
- Để vệt sáng ở tên Table cần nhập rồi chọn Open

Chú ý : trong quá trình nhập ta có thể điều chỉnh độ rộng các cột cho thích hợp bằng cách đưa dấu chuột về cạnh bên phải của tiêu đề cột cho xuất hiện dấu  $\Leftrightarrow$  rồi Drag chuột để điều chỉnh. Khi đóng ta lưu Layout bằng cách trả lời Yes

### 3. Hiệu chỉnh Table

Ta có thể hiệu chỉnh Table để : thay đổi cấu trúc bản ghi, sửa đổi nội dung bản ghi hoặc cách trình bày.

#### a. Thay đổi cấu trúc bản ghi

- Chọn tên của Table cần hiệu chỉnh.
- Chọn Design
- Hiệu chỉnh lại qua bảng :

Ta có thể thay đổi các thông tin liên quan đến các trường trong Table từ tên trường, kiểu, các thuộc tính, thêm bớt các trường...

#### b. Thay đổi nội dung bản ghi

- Chọn tên của Table cần hiệu chỉnh.
- Chọn Open hoặc Double Click tại đó.
- Hiệu chỉnh số liệu giống như trong Excel.

#### c. Thay đổi cách trình bày

- Chọn tên của Table cần hiệu chỉnh.
- Chọn Open hoặc Double Click tại đó.
- Chọn Format để định dạng, sau đó :
  - Font : thay đổi kiểu chữ.

- Cells : thay đổi cách thể hiện như : Gridlines Shown (che hay hiện đường lưới), Cell Effect (trình bày ô số liệu phẳng, nhô lên hoặc lõm xuống), Gridline Color (màu sắc của nét gạch), Background Color (màu nền của ô).
- Column Width : qui định độ rộng cột.
- Hide Column : che bớt cột. Nếu muốn hiện lại chọn Unhide Column.

#### **4. Khai thác số liệu trên Table**

Cho phép khai thác số liệu một cách tức thời khi đang làm việc trực tiếp trên Table. Nếu muốn tự động hóa công tác khai thác thông tin và có các báo cáo đẹp mắt thì ta phải dùng Report, Query, Macro hoặc lập trình bằng Visual Basic.

##### **a. Tìm và thay thế**

Cho phép tìm và thay thế nội dung trên một trường nào đó trong Table.

- Đưa con trỏ về trường cần tìm và thay thế.
- Chọn Edit - Replace

##### **b. Thay đổi vị trí trường**

- Chọn cột cần thay đổi vị trí (đưa dấu chuột lên tiêu đề trường).
- Drag chuột để đưa trường về vị trí mới.

##### **c. Sắp xếp**

- Chọn trường làm khóa để sắp xếp.
- Chọn trên thanh thực đơn Record - Sort (hoặc chọn biểu tượng)
- Chọn sắp tăng dần (Sort Ascending) hoặc giảm dần (Descending).

##### **d. Lọc bản ghi**

- Chọn trên thanh thực đơn Record - Filter (hoặc chọn biểu tượng)
- Chọn Filter by Form

Qui định cách lọc :

- Muốn lọc theo trường nào ta chỉ việc bấm chuột vào trường đó và chọn giá trị làm điều kiện để lọc.
- Bấm phím phải của chuột chọn Apply Filter. Lúc này chỉ còn các bản ghi thỏa mãn điều kiện.
- Nếu muốn hủy lọc thì bấm phím phải của chuột chọn Remove Filter. Lúc này hiện tất cả các bản ghi như ban đầu.

Chú ý : trong quá trình lọc ta có thể dùng các điều kiện với các phép toán so sánh và quan hệ.

## **IV. LÀM VIỆC VỚI QUERY**

### **1. Khái niệm**

Query là một công cụ cho phép người sử dụng thống kê số liệu, xây dựng các báo cáo tổng hợp dưới nhiều hình thức khác nhau trên dữ liệu gốc trong Table.



Muốn làm việc được với Query trước hết ta phải có Database và Table với dữ liệu nhập vào sẵn.

Query còn được dùng để tạo ra dữ liệu phục vụ cho các công cụ khác như Report, Form và cho cả một Query khác.

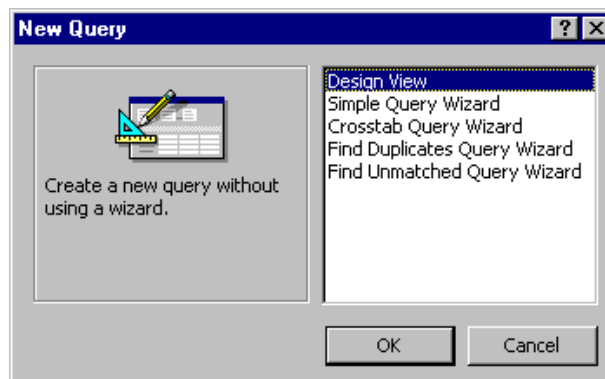
Tùy theo mục đích khai thác ta có thể sử dụng một trong các loại Query sau :

- Select Query : cho phép chọn lựa các bản ghi, tạo thêm các vùng tính toán và trả về kết quả là các bản ghi thỏa mãn điều kiện. Ta có thể dùng Query để thao tác trên nhiều Table cùng lúc.
- Append Query : nối thêm dữ liệu từ các bản ghi của một hay nhiều Table vào cuối một Table khác.
- Make-Table Query : tạo ra một Table mới từ một Dynaset (Dynamic Dataset). Cho phép tạo các Table dự phòng, trích bản ghi để lưu trữ trước khi xóa các bản ghi này khỏi Table hiện hành.
- Delete Query : xóa một nhóm các bản ghi từ một hay nhiều Table.
- Cross Tab Query : Query tham chiếu chéo, được dùng để tạo nhóm dữ liệu và trả về kết quả dưới dạng một bản tính kèm theo số cộng ngang, cộng dọc. Ta thường dùng loại này để tạo dữ liệu phục vụ cho các Report và Chart.
- Find Duplicate Query : tìm trong Table những bản ghi có giá trị giống nhau ở trên tất cả các trường.
- Find Unmatched Query : tìm những bản ghi mà giá trị của nó không trùng với giá trị của bất cứ một bản ghi nào trên một Table khác.
- Union Query : nối các bản ghi của hai hay nhiều Table thành một danh sách chung.
- Pass-Through Query : Query chuyển giao, dùng để gửi lệnh trực tiếp đến hệ ngôn ngữ SQL (Structured Query Language) của ACCESS.
- Data Definition Query : sử dụng các lệnh của ngôn ngữ SQL để tạo hoặc sửa đổi cấu trúc của một Table trong Database.

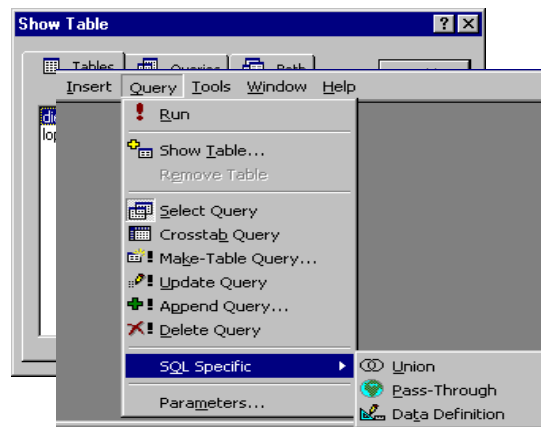
## 2. Cách tạo QUERY

Muốn tạo Query ta thực hiện qua các bước sau :

- Bước 1: trong hộp Database ta chọn nút Query , chọn New
- Bước 2: chọn kiểu tạo Query qua cửa sổ sau



- Bước 3: chọn Table phục vụ cho việc xây dựng Query qua cửa sổ sau :
  - Table : để xem danh sách tên các Table đã tạo trước đó.
  - Query : để xem danh sách các Query đã có.
  - Both : xem danh sách cả Table và Query.
  - Add : bổ sung Table hoặc Query được chọn cho việc tạo Query mới.
  - Close : đóng cửa sổ chọn.
- Bước 5: thiết kế Query theo yêu cầu. Nếu muốn thay đổi loại Query thì ta chọn trên thanh

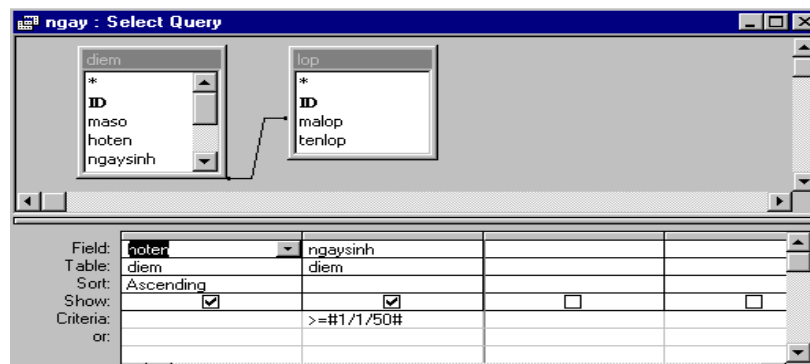


thực đơn chức năng Query sau đó chọn trong danh sách loại Query.

- Bước 6: tùy theo từng loại Query ta có cách thiết kế riêng. Sau khi tạo Query xong ta đóng lại và đặt tên cho Query khi máy yêu cầu.
- Sau đây ta xét cách tạo một sổ Query thường được sử dụng (chỉ làm rõ cho bước 6).

#### a. *Select Query*

Sau khi chọn 5 bước trên và loại Query là Select Query màn hình sẽ xuất hiện cửa sổ khai báo Query như sau :



- Field : chọn tên các trường có liên quan đến điều kiện, sắp xếp hoặc cần xem.
- Table : hiển thị tên của Table mà trường được chọn trực thuộc vào nó.
- Sort : qui định việc sắp xếp tăng (Ascending) hoặc giảm (Descending) theo nó.
- Show : cho phép hiển thị nội dung của trường hay không ( : không, √ : có).

- Criteria : qui định điều kiện để lọc các bản ghi cần ghi. Nếu các điều kiện viết trên cùng dòng này thì ngầm định là quan hệ AND nếu viết trên dòng phía dưới thì quan hệ OR

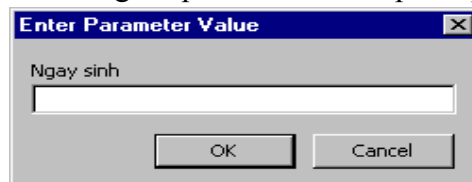
Đóng cửa sổ Select Query và ghi tên của Query cần lưu trữ lên đĩa.

Chú ý :

- Để linh hoạt khi thay đổi điều kiện của Query trong mục Criteria thay vì gõ giá trị làm điều kiện ta nhập vào tên biến nhớ. Lúc đó mỗi khi gọi Query máy sẽ yêu cầu nhập vào giá trị tương ứng.

Ví dụ : muốn xem danh sách sinh viên sinh vào một ngày nào đó ta đưa con trỏ về ô Criteria của này sinh gõ [ngay sinh]. Khi gọi Query thực hiện sẽ xuất hiện cửa sổ hỏi ngày sinh :

- Nếu muốn tính tổng theo từng bộ phận thì ta bấm phím phải của chuột và chọn Total, lúc



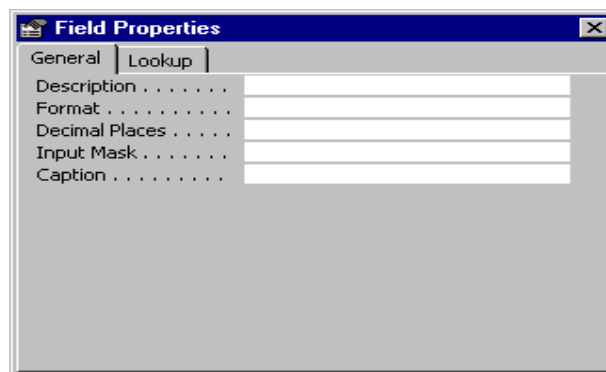
đó trong Query xuất hiện một dòng Total để ta qui định phương thức tính toán.

Khi bấm vào Total ta thấy xuất hiện các hàm tính toán : Sum (tính tổng), Avg (tính giá trị trung bình), Min (tìm giá trị nhỏ nhất), Max (tìm giá trị lớn nhất), Count (đếm số lượng các giá trị), StDev (độ lệch chuẩn của các giá trị), Var (sự biến thiên của các giá trị) và các tùy chọn khác là : Group By (định nghĩa nhóm muốn tính toán), Expression (tạo ra một biểu thức tính toán), Where (chỉ định điều kiện khi tính toán).

- Nếu muốn tạo ra một trường mà nội dung của nó được tính toán từ một biểu thức bất kỳ thì ta đưa con trỏ đến cột đó viết biểu thức cần tính.

Cách viết : Tên trường : biểu thức. Ví dụ : DTB:([mon1+[mon2])/2

- Nếu muốn thay đổi thêm cho cột cần thể hiện thì ta đưa dấu chuột đến cột đó nhấn phím phải của chuột làm xuất hiện thực đơn rồi chọn Properties và qui định qua :



- Description : dòng chú thích.
- Format : qui định khuôn dạng cách thể hiện nội dung dữ liệu.
- Decimal Places : số chữ số ở phần thập phân.
- Input Mask : cách thức nhập số liệu.

- Caption : tiêu đề sử dụng thay cho tên trường.

### b. Cross Tab Query

Đây là loại Query cho phép lập bảng tham chiếu chéo : tổng hợp từ một đến nhiều chỉ tiêu theo hàng, trên mỗi hàng lại tổng hợp một chỉ tiêu khác theo cột, vùng giao nhau giữa hàng và cột thể hiện trị số tổng hợp của một chỉ tiêu thứ ba.

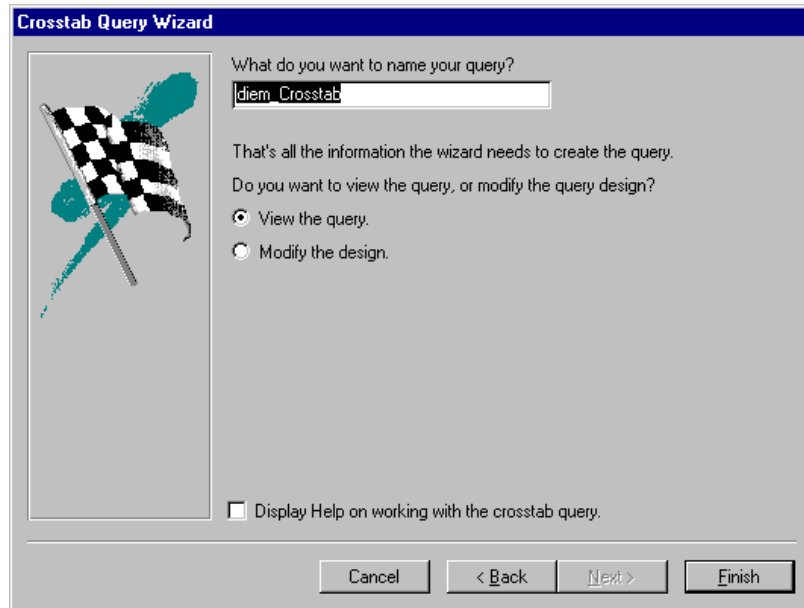
Chọn Cross Tab Query, sau đó chọn tên các Table chứa số liệu để lập Query hoặc tên Query cơ sở để tạo Query kế tiếp. Sau đó chọn Next để xuất hiện cửa sổ :

- Trong Available Field ta chọn tên trường làm tiêu đề dòng. Số trường tối đa được chọn là 3.
- Select Filed : chứa tên các trường được chọn.

- Chọn Next để xuất hiện màn hình kế tiếp :

- Chọn tên trường sử dụng làm tiêu đề cột.

- Chọn Next để sang bước kế tiếp.
- Chọn giá trị số cần tính tại mỗi giao điểm dòng và cột.
- Chọn Next để chuyển sang bước kế tiếp.
- Gõ vào tên của Query cần tạo.
- Chọn Finish để hoàn tất tạo Query.



- Chú ý : tương tự tạo các Query còn lại.

### 3. Hiệu chỉnh QUERY

Nếu muốn hiệu chỉnh lại Query thì ta đưa con trỏ về tên của Query và chọn Design để thực hiện thiết kế lại.

Nếu muốn xem lệnh tạo Query thì trong quá trình Design ta bấm nút phải của chuột rồi chọn SQL View.

### 4. Thực hiện QUERY

Đưa con trỏ về tên của Query và chọn Open ( hoặc Double Click chuột tại đó).

## V. Làm việc với Report

### 1. Khái niệm

Report cho phép người sử dụng thiết kế các mẫu báo cáo theo yêu cầu để xem và in ấn các báo cáo đó ra giấy.

Report là một công cụ rất mạnh để chúng ta có thể tạo ra một báo cáo đẹp mắt. Ta dễ dàng điều chỉnh kích cỡ, kiểu dáng của mọi thành phần trong Report. Đa số các thông tin trong Report được lấy từ các Table, Query, các lệnh của SQL..

Có hai dạng Report chính là :

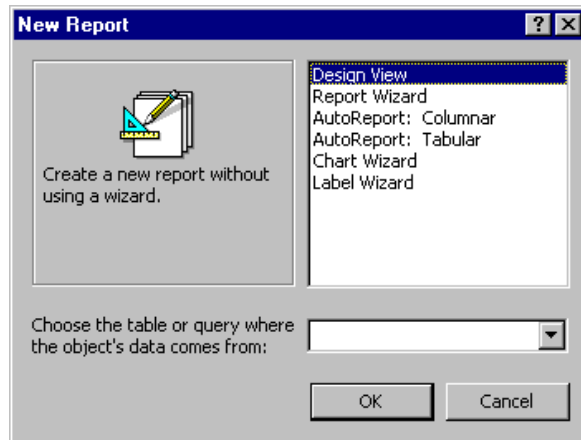
- Columnar Report : báo cáo dạng cột. Thường sử dụng để in các phiếu theo mẫu cho trước. Ví dụ : in lại phiếu thu, phiếu chi, phiếu báo điểm...

- Tabular Report : báo cáo dạng bảng. Đây là loại thường sử dụng để in các bảng kê. Trong trường hợp này các trường bố trí trên cột đứng, bản ghi bố trí trên dòng ngang.

## 2. Cách tạo Report

Bước 1 : chọn vào nút Report, tiếp đến chọn New

Bước 2 : chọn phương pháp và loại Report cần tạo qua cửa sổ :



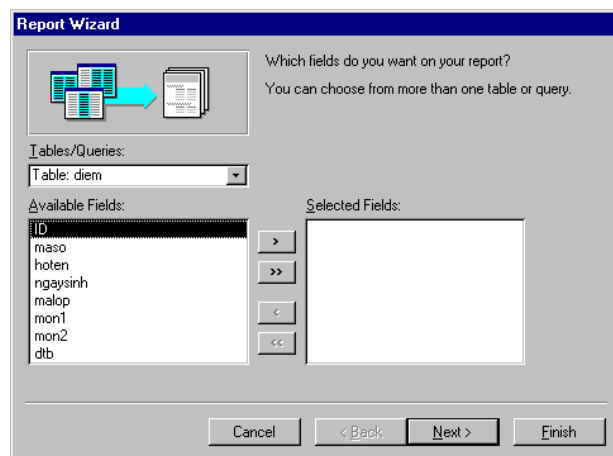
- Choose the table or query where the object's data comes from : chọn tên của Table hoặc Query chứa số liệu cơ sở của Report.
- Design View : tự thiết kế Report từ màn hình trắng.
- Report Wizard : thiết kế Report với sự trợ giúp của ACCESS.
- AutoReport - Columnar : tự động tạo báo cáo dạng cột.
- AutoReport - Tabular : tự động tạo báo cáo dạng bảng.
- Chart Wizard : tạo đồ thị mô tả với sự trợ giúp của ACCESS
- Label Wizard : tạo nhãn với sự trợ giúp của ACCESS.

Chọn OK để chuyển sang bước kế tiếp khai báo Report.

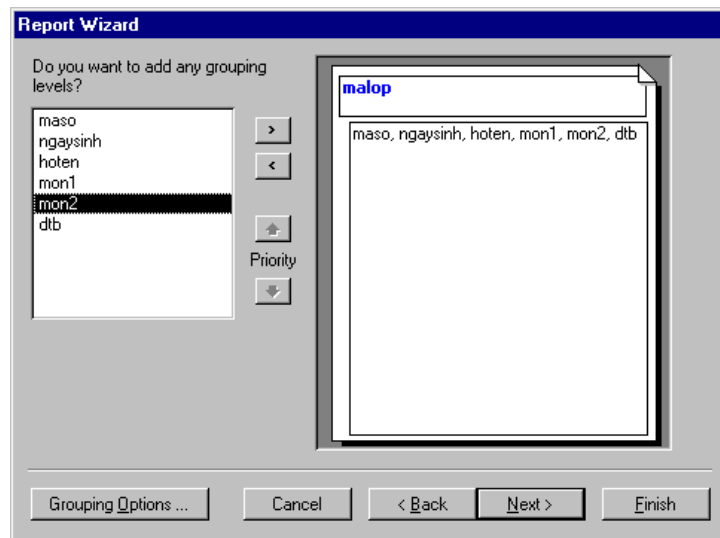
Sau đây giới thiệu phần thiết kế Report theo các kiểu chọn trên :

### Report Wizard :

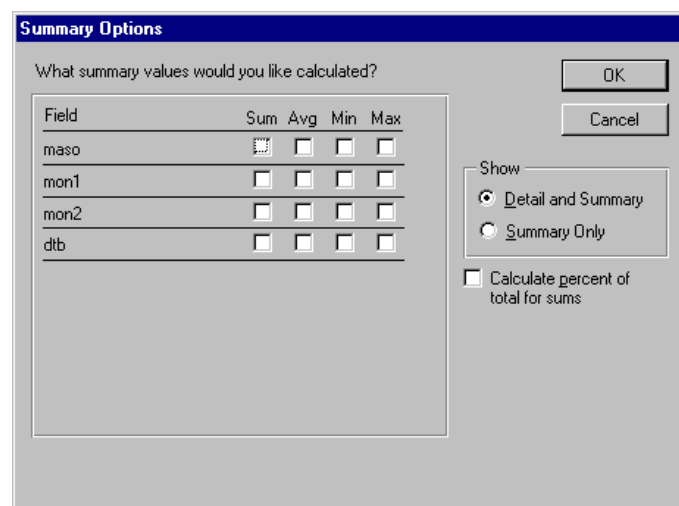
Sau khi chọn Report Wizard xuất hiện cửa sổ để chọn tên các trường có nội dung cần xem trong Report từ Available Fields để chuyển sang Selected Fields.



- Chọn Next chuyển sang bước tiếp theo.
- Chọn tên trường cần dồn nhóm theo nó. Nếu không muốn kết nhóm thì bỏ qua bước này bằng cách chọn Next. Ví dụ : ta muốn in danh sách sinh viên trong trường theo từng nhóm là lớp thì chọn trường nhóm là lớp.
- Grouping Option : qui định thêm thông tin về phương pháp tạo nhóm.
- Chọn Next để chuyển sang bước tiếp theo :

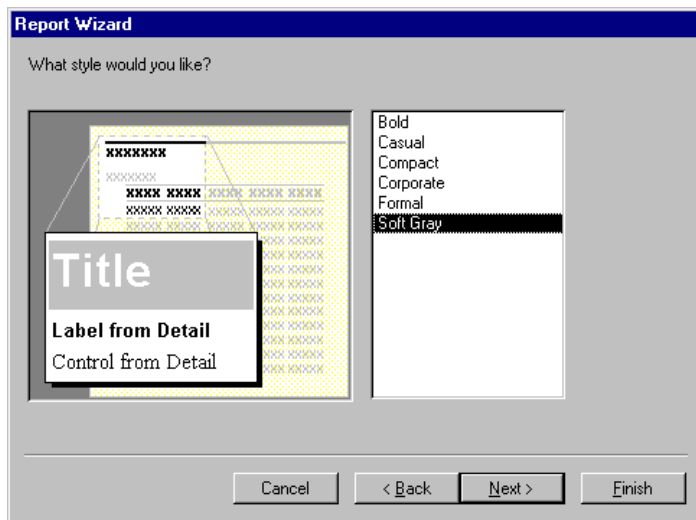


- You can sort records by up to four fields, in either ascending or descending order : qui định việc sắp xếp bản ghi theo thứ tự tăng hoặc giảm tối đa theo bốn khóa. Nếu muốn sắp xếp thì ta chọn tên trường khóa và qui định tăng hoặc giảm.
- Summary Option : Cho phép thực hiện tính toán theo các trường. Lúc đó xuất hiện mẫu khai báo :



Lúc này ta phải qui định tên trường và công thức tính toán cho từng trường đó. Trong Table có bao nhiêu trường kiểu số thì có bấy nhiêu trường có thể được tính toán. Trong mục Show nếu chọn Detail and summary sẽ có các dòng chi tiết lẫn các dòng tóm tắt, nếu chọn Summary Only thì chỉ có các dòng tính tổng. Nếu chọn Calculate Percent thì sẽ có tính tính tỉ lệ phần trăm.

- Sau đó chọn Next để tiếp tục.



Qui định cách trình bày Report theo một trong 6 kiểu trên mục Layout. Mục Orientation cho phép qui định cách bố trí theo chiều ngang hay dọc của trang giấy.

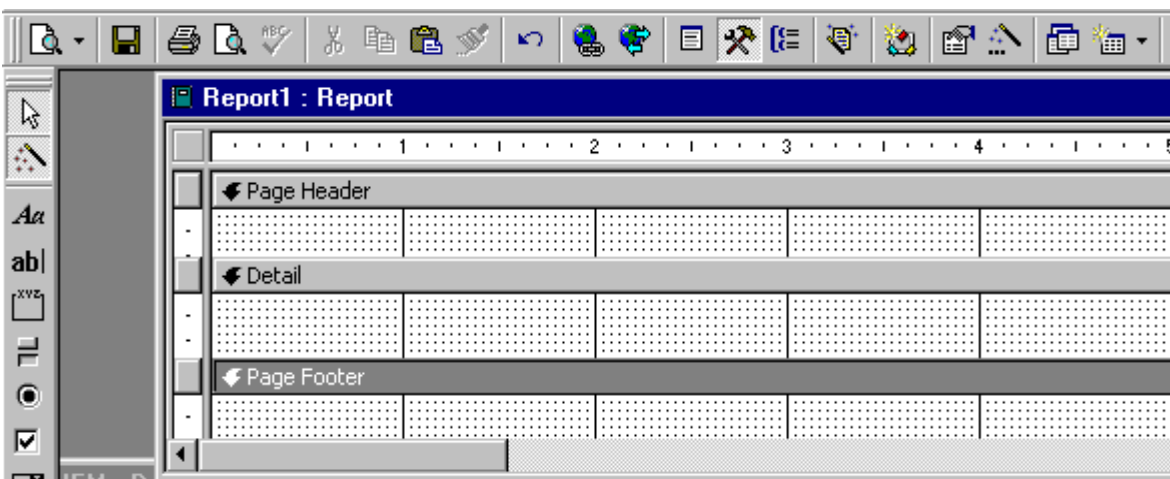
Qui định cách trình bày tiêu đề. Sau đó chọn Next để hoàn tất việc khai báo. Lúc này khai báo tên của Report để ghi vào đĩa và chọn Finish để hoàn tất.

**AutoReport Columnar** : tự động tạo ra một báo cáo dạng cột từ một Table cho trước.

**AutoReport Tabular** : tự động tạo ra một báo cáo dạng bảng từ một Table cho trước

**Design View** : cho phép người sử dụng tự thiết kế một Report từ đầu đến cuối.

- Chọn tên của Table hoặc Query chứa dữ liệu cơ sở rồi chọn kiểu tự thiết kế Design View. Lúc đó trên màn hình xuất hiện cửa sổ cho phép thiết kế Report như sau :



- Page Header, Page Footer : cho phép ghi nội dung tiêu đề đầu hoặc cuối mỗi trang in.
- Detail : ghi nội dung của các dòng trong Report.
- Nếu muốn tạo tiêu đề được in một lần ở đầu báo cáo ta chọn trên thanh thực đơn Cột bên trái xuất hiện thanh công cụ để phục vụ người dùng thiết kế Report, ý nghĩa của chúng như sau :





Select Object : cho phép chọn đối tượng.  
 Control Wizard : thiết kế với giúp đỡ của ACCESS  
 Label : tạo nhãn  
 Textbox : hộp dữ liệu  
 Option Group : tạo một Frame  
 Toggle Button : tạo nút  
 Option Button : nút chọn  
 Check Box : hộp kiểm tra

Ngoài ra còn có nh

### 3. Hiệu chỉnh Report

Sau khi đã thiết kế và lưu trữ Report, nếu muốn hiệu chỉnh lại Report thì:

- Trong hộp Database đưa con trỏ đến tên Report cần hiệu chỉnh.
- Chọn Design.
- Hiệu chỉnh giống như trong Design View.

### 4. Thực hiện Report

Nếu muốn xem hoặc in Report ta chọn trên thanh thực đơn :

- File → Print (để in) hoặc Print Preview (để xem)

Hoặc bấm vào các biểu tượng tương ứng trên thanh công cụ

## VI. Làm việc với Form

### 1. Khái niệm

Form là môth công cụ cho phép người sử dụng thiết kế các mẫu nhập và xem số liệu giống như mẫu biểu ngoài thực tế.

Ta thấy khi nhập liệu bằng Table thì tuy việc nhập liệu rất dễ dàng nhưng các mẫu nhập của nó khác nhiều với trong thực tế tạo cảm giác khó chịu khi chưa quen. Vì vậy, khi xây dựng chương trình cho nhiều người sử dụng đặc biệt là những người không chuyên dùng máy tính thì ta phải tạo ra các mẫu biểu giống hệt trong thực tế để dễ dàng khi sử dụng.

Dữ liệu sử dụng trong Form được lấy từ Query hoặc Table

### 2. Thiết kế Form

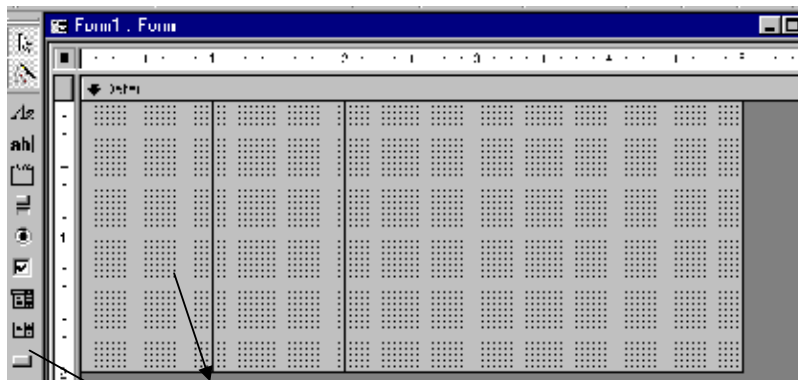
Ta có thể tạo Form bằng nhiều cách khác nhau như :

- Dùng AutoForm : tự động tạo Form từ Table hoặc Query cho trước.
- Dùng Wizard : tạo Form qua sự giúp đỡ của ACCESS.
- Tự thiết kế Form

Trên thực tế để tạo Form một cách nhanh chóng, dễ dàng và đẹp mắt ta thường sử dụng Form Wizard để tạo ra mẫu Form và sau đó tự hiệu chỉnh lại những chỗ cần thiết.

## Tự thiết kế Form

- Chọn tên Table hoặc Query, chọn kiểu thiết kế Form là Design View rồi OK.
- Xuất hiện màn hình để thiết kế Form như sau :



Màn hình chứa nội dung của Form

Thanh công cụ : chứa các công cụ để phục vụ cho việc thiết kế.

Tên gọi và ý nghĩa của các nút chọn trên thanh công cụ theo thứ tự từ trái sang phải như sau:

- Select : chọn đối tượng cần hiệu chỉnh.



- Control Wizard: trợ giúp của Wizard khi tạo Control.
- Label : tạo nhãn.
- TextBox: nội dung biểu thức hoặc trường.
- Option Group: nhóm chọn việc.
- Toggle Button: tạo nút bật tắt.
- Option Button: tạo nút chọn một trong nhiều giá trị.
- CheckBox: hộp đánh dấu để chọn nhiều giá trị cùng lúc.
- ComboBox: hộp chọn cặp.
- ListBox: hộp xem, chọn trong một danh sách.
- Command Button: tạo nút lệnh.
- Image: tranh ảnh.
- Unbound Object Frame: tạo một khung hình cố định.
- Bound Object Frame: tạo khung hình không cố định.
- Page Break: tạo dấu phân trang.
- Tab Control: tạo Tab điều khiển để chọn trang.
- SubForm: tạo Form con.
- Line: vẽ đường thẳng.

- Rectangle: vẽ hình chữ nhật.
- More Control: chọn sử dụng các nút điều khiển từ nhiều chương trình khác.

Muốn đưa một công cụ vào trong Form ta có thể tự thiết kế nút đó hoặc sử dụng Control Wizard của ACCESS :

a. ***Nếu dùng Control Wizard***

- Bấm nút Control Wizard (cho nó có màu sáng).
- Bấm vào công cụ cần chọn để đưa vào Form.
- Drag chuột vào trong Form tại khu vực cần đặt công cụ đó.
- Khai báo các thông tin cần thiết theo chỉ dẫn của ACCESS.

b. ***Tự thiết kế***

- Bấm chuột vào công cụ cần chọn để đưa vào Form.
- Drag chuột vào trong Form tại khu vực cần đặt công cụ đó.
- Khai báo các thông tin cần thiết. Nếu muốn sửa đổi các thuộc tính thì bấm đúp chuột tại công cụ vừa tạo để khai báo lại.

**3. Hiệu chỉnh Form**

- Chọn tên của Form cần hiệu chỉnh, chọn Design
- Sửa đổi giống như trong phần tự thiết kế.

**4. Thực hiện Form**

- Chọn tên của Form.
- Chọn Open

## **VII. Macro và hệ thống thực đơn**

### **1. Macro**

a. ***Khái niệm :***

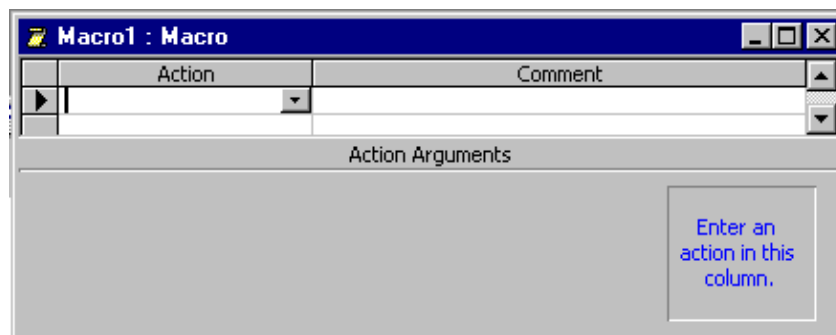
Macro là một hay một tập hợp các hành động (Action) liên tiếp được định nghĩa và lưu trữ với một tên xác định. Macro cho phép tự động hóa các công việc cần thực hiện.

Có ba loại Macro chính là :

- Macro kết hợp nhiều hành động : là Macro được kết hợp bởi nhiều hành động liên tiếp nhau. Khi tên Macro được gọi các hành động này sẽ lần lượt được tự động thực hiện.
- Macro Group : là một tập hợp các Macro có các tính năng giống nhau. Nó cho phép quản lý cơ sở dữ liệu dễ dàng hơn. Để thi hành một Macro trong Macro Group ta chỉ tên của nó như sau : Tên Macro Group.Tên Macro thực hiện.
- Macro theo điều kiện : là Macro mà các hành động chỉ được thi hành khi thỏa mãn điều kiện nào đó. Điều kiện là một biểu thức được chỉ định trong Condition.

b. ***Cách tạo Macro***

- Bước 1: trong cửa sổ Database chọn nút Macro, tiếp đến chọn New
- Bước 2 :xuất hiện cửa sổ để khai báo Macro như sau :
- Trong Action ta chọn một hành động cần thực hiện. Ta có thể chọn nhiều hành động tương



ứng với nhiều dòng.

- Trong cột Comment ta có thể ghi rõ chú thích về hành động. Cột này không bắt buộc nhưng nó giúp người sử dụng dễ dàng khi bảo trì hệ thống vì biết được ý đồ thực hiện khi thiết kế.
- Trong mục Action Arguments ta có thể chỉ định các đối số cho Action nếu cần thiết.

### c. Thực hiện Macro

Để thực hiện Macro ta có thể chọn tên của Macro trong Database rồi chọn tiếp Open

Hoặc gọi tên Macro trong khi sử dụng Form, Report...

## 2. Hệ thống thực đơn

Ta có thể sử dụng Macro để xây dựng hệ thống thực đơn cho phép lựa chọn công việc một cách dễ dàng và tiện lợi. Thông qua hệ thống thực đơn ta có thể liên kết tất cả các đối tượng trên Database thành một hệ thống chương trình thống nhất tiện lợi cho người sử dụng chương trình.

### a. Cách tạo thực đơn

Giả sử ta muốn tạo một hệ thống thực đơn gồm các mục như sau :

Mục 1	Mục 2	Mục 3
Mục 1-1	Mục 2-1	Mục 3-1
Mục 1-2	Mục 2-2	Mục 3-2
...	...	...
Mục 1-n	Mục 2-n	Mục 3-n

Trong hệ thống thực đơn này các mục nằm ngang gọi là Menu cấp 1, mỗi cột đứng là một Menu cấp 2 (ta có 3 Menu cấp 2) và tương tự có thể tạo Menu các cấp thấp hơn (Ví dụ : chọn vào Mục 1-1 thì xuất hiện các mục Mục 1-1-1, Mục 1-1-2...).

### Bước 1: tạo menu cấp 1.

- Bấm dấu chuột vào nút Macro, chọn New.

- Khai báo vào bảng sau :
- Action : lựa chọn hành động là AddMenu cho cả ba

Action	Comment
AddMenu	Muc 1
AddMenu	Muc 2
AddMenu	Muc 3

Menu Name: &Muc 1  
Menu Macro Name: Muc1  
Status Bar Text:

Enter the name of the menu as you want it to appear on the custom menu bar (for example, File or Edit). Required argument for custom menu bars and global menu bars, but ignored

- Comment : ghi dòng chú thích. Mục này không cần.
- Menu Name : ghi nội dung dòng chữ sẽ hiện trên thanh thực đơn. Trong trường hợp này ta đặt tên là : Muc 1, Muc 2, Muc 3. Nếu muốn xuất hiện dấu gạch chân dưới chữ cái dùng làm phím nóng thì thêm vào trước chữ &
- Menu Macro Name : tên của Macro. Ta phải nhớ tên này để sau này gọi lại trong khi tạo menu cấp 2. Trong trường hợp ta đặt tên các Macro là : Muc1, Muc2, Muc3
- Status Bar Text : nội dung dòng chữ sẽ xuất hiện trên thanh Menu Bar khi ta chọn vào mục này.
- Ta đóng cửa sổ này bằng cách bấm chuột vào góc trên bên phải nơi có dấu X và đặt tên cho Macro là MainMenu (Tên này ta tự qui định).

## Bước 2: tạo các menu cấp 2.

- Vào hộp Database chọn nút Macro rồi chọn New.
- Xuất hiện cửa sổ giống bước 1, ta chọn thêm View - Macro Name, sẽ xuất hiện cửa sổ mới như sau :

Macro Name	Action	Comment
Muc 1-1	OpenForm	
Muc 1-2		
Muc 1-3		
...		
Muc 1-n		

Form Name: NHAPDIEM  
View: Form  
Filter Name:  
Where Condition:  
Data Mode:  
Window Mode: Normal

Opens a form in Form view, Design view, Print Preview, or Datasheet view. Press F1 for help on this action.

- Macro Name : gõ vào tên các mục trên Menu cấp hai thứ nhất. Những chữ này sẽ được in ra trên thanh thực đơn.

- Action : hành động cần thực hiện khi ta chọn vào chức năng này. Ta chọn các Action này trong danh sách mà ACCESS cho trước.
- Action Argument : khai báo các tham số liên quan đến Action.
- Đóng cửa sổ này và gõ vào tên của Macro để lưu trữ lên đĩa. Tên này phải trùng với tên của Macro (mà ta đã khai báo trong Menu Macro Name ở bước1). Trong trường này ta gõ tên là Muc1.
- Tương tự, ta tạo hai Macro cấp 2 khác và đặt tên là Muc2, Muc3

**Bước 3:** gắn Menu lên một Form hoặc Report.

- Trong cửa sổ Database chọn Form (hoặc Report). Chọn New. Bấm chuột vào hộp Properties trên thanh Menu Bar để xuất hiện hộp thoại :

Property	Value
Pop Up	No
Modal	No
Cycle	All Records
Menu Bar	MAINMENU
Toolbar	
Shortcut Menu	Yes
Shortcut Menu Bar	
Fast Laser Printing	Yes
Help File	
Help Context Id	0
Tag	
Has Module	No

**b. Sử dụng thực đơn**

Khi nào muốn dùng thực đơn chọn viện ta chỉ việc mở Form có gắn với thanh thực đơn được tạo.

## CHƯƠNG 3. BẮT ĐẦU LẬP TRÌNH VỚI VISUAL BASIC

### I. Giới thiệu

Ngôn ngữ BASIC (Beginner's All Purpose Symbolic Instruction Code) đã có từ năm 1964. BASIC rất dễ học và dễ dùng. Trong vòng 15 năm đầu, có rất nhiều chuyên gia tin học và công ty tạo các chương trình thông dịch (Interpreters) và biên dịch (Compilers) cho ngôn ngữ này và đã góp phần làm cho BASIC trở nên rất phổ dụng.

Năm 1975, Microsoft tung ra thị trường sản phẩm đầu tay Microsoft BASIC và tiếp đó Quick BASIC (còn gọi là QBASIC) thành công rực rỡ. Quick BASIC phát triển trong nền Windows nhưng vẫn khó khăn khi tạo giao diện kiểu Windows. Sau đó nhiều năm, Microsoft bắt đầu tung ra một sản phẩm mới cho phép chúng ta kết hợp ngôn ngữ dễ học BASIC và môi trường phát triển lập trình với giao diện người dùng đồ họa (Graphic User Interface - GUI) trong Windows. Đó là Visual Basic Version 1.0 vào năm 1991.

Trước đó, chúng ta không có một giao diện người dùng đồ họa (GUI) với một IDE (Integrated Development Environment – môi trường phát triển tích hợp) để giúp các chuyên gia lập trình phát triển chương trình dễ dàng và có thể tập trung công sức và thời gian vào các vấn đề liên quan đến tổ chức bài toán. Mỗi người phải tự thiết kế giao diện qua thư viện có sẵn Windows API (Application Programming Interface) trong nền Windows. Điều này tạo ra những trở ngại không cần thiết làm phức tạp việc lập trình.

Visual Basic giúp chúng ta bỏ qua những khó khăn đó, các chuyên gia lập trình có thể tự thiết kế cho mình giao diện cần thiết trong ứng dụng (application) một cách dễ dàng và như vậy có thể tập trung nỗ lực để giải quyết các vấn đề cần giải quyết trong doanh nghiệp hay kỹ thuật.

Ngoài ra, còn nhiều công ty phụ phát triển thêm các thủ tục, hàm (modules), công cụ (tools, controls) hay ứng dụng (application) hỗ trợ dưới hình thức VBX cộng thêm vào giao diện chính nên VB càng lúc càng thêm phong phú.

Khi Visual Basic phiên bản 3.0 được giới thiệu, thế giới lập trình lại thay đổi lần nữa. Với phiên bản này, chúng ta có thể thiết kế các ứng dụng (application) liên quan đến cơ sở dữ liệu tương tác đến người dùng qua DAO (Data Access Object). Các ứng dụng loại này thường gọi là ứng dụng ngoại vi (front-end application).

Phiên bản 4.0 và 5.0 mở rộng khả năng VB nhắm đến hệ điều hành Windows 95.

Phiên bản 6.0 cung ứng một phương pháp mới để kết nối với cơ sở dữ liệu qua sự kết hợp của ADO (Active Data Object). ADO còn giúp các chuyên gia phát triển mạng nối với cơ sở dữ liệu khi dùng Active Server Pages (ASP).

Lưu ý rằng, tất cả các khái niệm và công dụng của Modules, Tools, Controls, DAO, ADO hay ASP sẽ được trình bày trong các bài học sau. Tuy nhiên, VB phiên bản 6.0 (VB6) không cung ứng tất cả các đặc trưng của kiểu mẫu ngôn ngữ lập trình hướng đối tượng (Object Oriented Language - OOL) như các ngôn ngữ C++, Java.

Visual Basic là một ngôn ngữ lập trình trực quan và thường được sử dụng hiện nay. Giống như các ngôn ngữ khác, khi lập trình ta buộc phải tuân theo các qui tắc, trình tự lô-gíc nhất

định nhưng nếu so với các ngôn ngữ lập trình có cấu trúc như Turbo Pascal, C... thì Visual Basic đi theo một phương pháp lập trình mới. Visual Basic xây dựng một môi trường làm việc dưới dạng các biểu mẫu (Form), các hộp điều khiển (Control Box), thiên về các đối tượng (Object oriented), những thủ tục được xử lý theo tình huống và các phương thức (Method).

Khi làm việc với Visual Basic người lập trình có nhiệm vụ chính là thiết kế biểu mẫu, các khung giao diện, các nút lệnh và công việc sẽ thực hiện tương ứng trên đó; các lệnh, các chỉ thị phải được viết ra sẽ hạn chế tối đa.

Một trong những điểm khác biệt rõ ràng nhất giữa Visual Basic và các ngôn ngữ lập trình có cấu trúc là một ngôn ngữ xử lý theo tình huống (event - driven language) và một ngôn ngữ xử lý theo thủ tục (procedural - language).

Đối với các ngôn ngữ xử lý theo thủ tục thì một chương trình ứng dụng sẽ cho thi hành một cách lô-gíc theo từng lệnh một tùy theo cách sắp xếp, tổ chức của người viết chương trình. Còn ngôn ngữ xử lý theo tình huống thì các chỉ thị chương trình chỉ được thực hiện khi gặp một tình huống đặc biệt xảy ra. Mỗi một tình huống tương ứng một thủ tục được thực hiện và các thủ tục này trong chương trình là hoàn toàn độc lập.

## II. Các khái niệm thường dùng

**Đối tượng (Object)** : là một tập hợp bao gồm chương trình và dữ liệu liên quan với nhau tạo thành một đơn vị xử lý độc lập. Khi khai báo đối tượng xong ta chỉ cần truyền cho nó các tham số cần thiết khi muốn đối tượng hoạt động. Khi đã hoàn tất khai báo, thực hiện thử để kiểm tra ta có thể lưu trữ đối tượng để sử dụng trong các chương trình khác. Trong Visual Basic các đối tượng chính là biểu mẫu (FORM) và hộp điều khiển (CONTROL). Mỗi một đối tượng ta có thể khai báo cho nó một số các thuộc tính riêng như màu sắc, kích thước, giá trị... và các thuộc tính này có thể thay đổi trong quá trình thực hiện chương trình.

**Biểu mẫu (Form)** : là một khung cửa sổ hiện trên màn hình và nó có thể chứa một dãy các hộp điều khiển trên đó. Tất cả các dữ liệu muốn nhập, xem đều được trình bày trên biểu mẫu này.

**Hộp điều khiển (Control Box)** : là một đối tượng đặt trên Form, mỗi một hộp điều khiển sẽ tương ứng với một chức năng nào đó sẽ được thực hiện.

**Thủ tục tình huống (Event procedure)** : là một dãy các chỉ thị lệnh và sẽ được tự động thực hiện khi xảy ra tình huống tương ứng. Một đối tượng có thể bao gồm nhiều thủ tục tình huống như vậy.

**Phương thức (Method)** : là các lệnh thao tác lên một đối tượng để thực hiện các xử lý theo yêu cầu nào đó (giống như một chương trình con). Mỗi phương thức sẽ mang một tên xác định và nếu muốn thực hiện phương thức ta viết như sau : <tên đối tượng>.<tên phương thức>[tham số]

Ví dụ : *Form1.Print "In lên màn hình"*, trong đó đối tượng là *Form1*, phương thức là *Print* và tham số là nội dung *"In lên màn hình"*.

## III. Lập trình trong Visual Basic

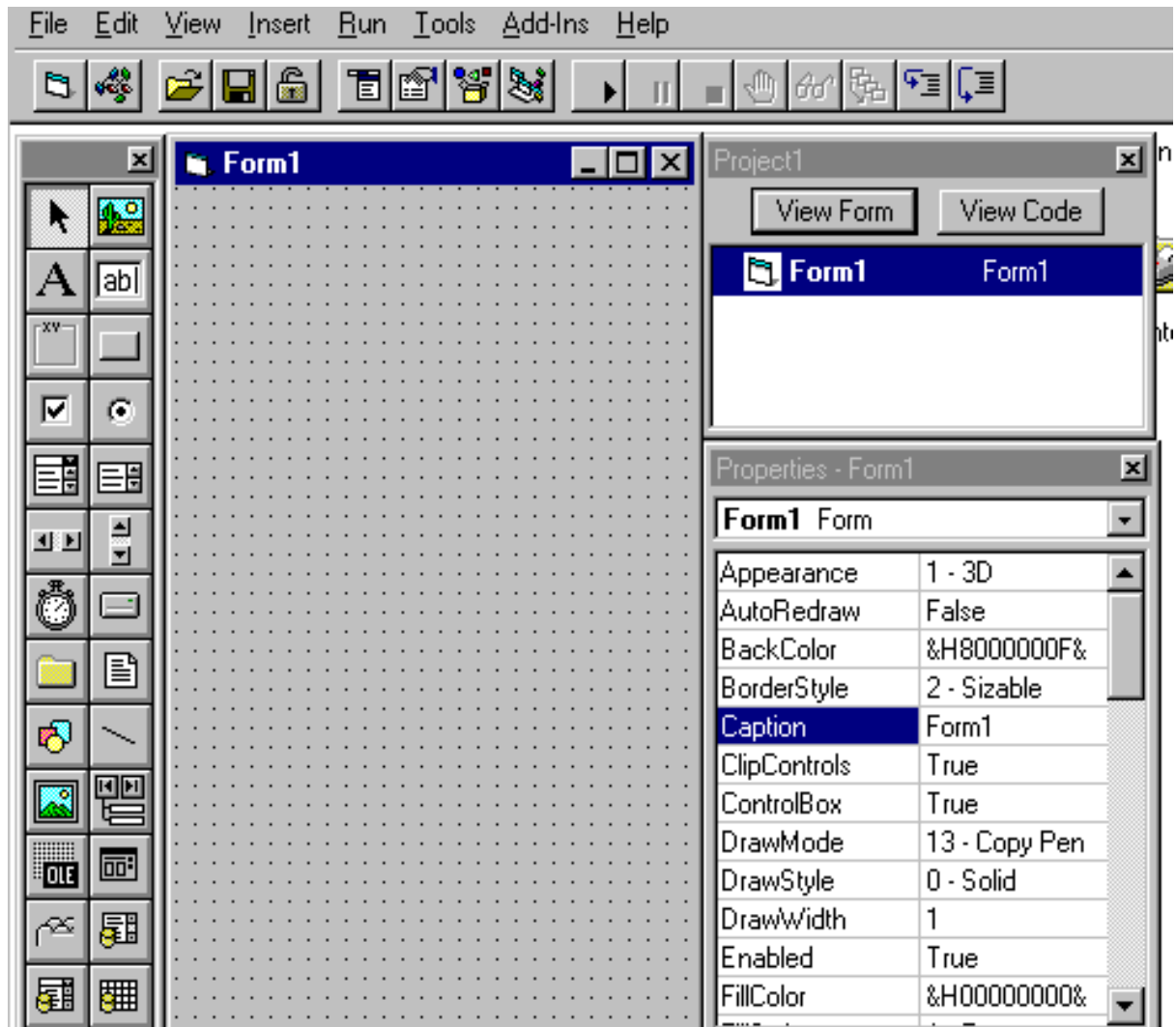
Khi lập trình trong Visual Basic, mọi ứng dụng đều thường bắt đầu bằng biểu mẫu (Form), đây là nơi hiển thị tất cả các thông tin liên quan đến ứng dụng.



Khi thiết kế chương trình ta thường qua các bước :

- Gắn các hộp điều khiển vào biểu mẫu.
- Thay đổi thuộc tính của hộp điều khiển.
- Viết thủ tục tình huống cho hộp điều khiển đó.

Trong phần này ta sẽ giới thiệu tổng quát về các bước trên.

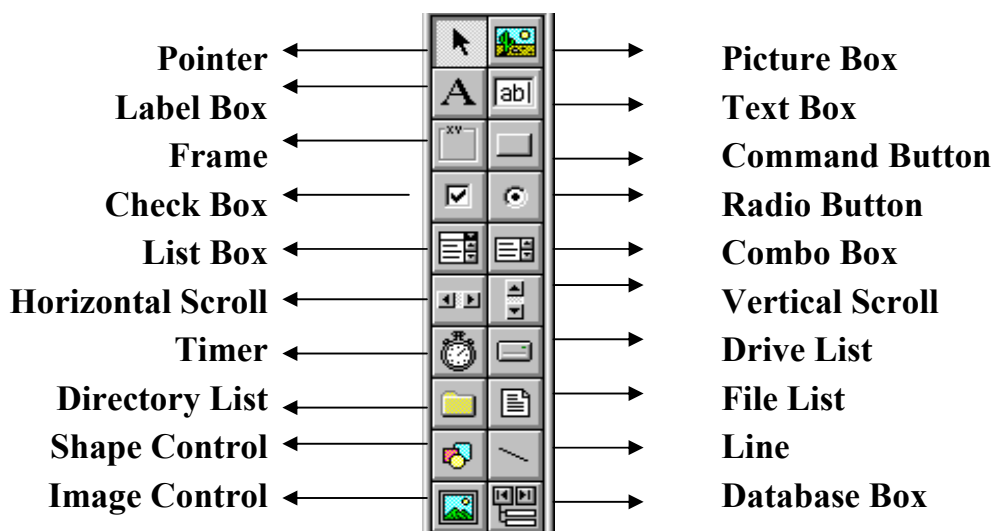


## 1. Làm việc với hộp/nút điều khiển

Hộp điều khiển là thành phần cơ bản nhất của biểu mẫu, nó quyết định hình dạng và hoạt động của ứng dụng. Một ứng dụng có linh hoạt, cung cấp đủ tiện nghi cho người dùng hay không sẽ phụ thuộc phần lớn vào việc sử dụng và khai thác các hộp điều khiển.

### a. Các loại hộp điều khiển :

Trên thanh Tools Bar có các hộp điều khiển thường sử dụng như :



Ý nghĩa các hộp điều khiển như sau :

Tên gọi	Ý nghĩa
Pointer	Di chuyển điểm đặt
Label Box	Ghi nội dung dòng văn bản. Nội dung cố định.
Frame	Tạo khung hình chữ nhật chứa nhiều hộp điều khiển
Check Box	Dùng chọn giá trị True hoặc False, nhiều hơn một mục
List Box	Liệt kê các giá trị để chọn lựa
Horizontal Scroll	Thanh cuộn ngang để thay đổi phần màn hình cần xem
Timer	Dùng bày tính hướng theo thời gian. Kiểm tra quá hạn
Directory List	Liệt kê tên các thư mục
Shape Control	Cung cấp công cụ để vẽ hình.
Image Control	Hiển thị hình ảnh trong tập tin Bitmap
Picture Box	Hiển thị hình ảnh đồ họa bất kỳ. Dung lượng lớn hơn Image
Text Box	Giữ nội dung văn bản. Có thể sửa đổi.
Command Button	Nút lệnh để chọn thực hiện một lệnh nào đó
Radio Button	Ô đài. Cho phép chọn một trong nhiều giá trị.
Combo Box	Ô hỗn hợp, phối hợp hộp văn bản và liệt kê
Vertical Scroll	Thanh cuộn đứng
Drive List	Liệt kê tên các ổ đĩa trên máy
File List	Liệt kê tên tập tin
Line	Cho phép vẽ đường thẳng
Database Box	Cho phép thao tác với cơ sở dữ liệu

### **b. Thêm hộp điều khiển lên biểu mẫu**

Để bổ sung một hộp điều khiển lên biểu mẫu ta có thể thực hiện theo một trong ba cách sau:

#### **Cách 1 :**

- Click chuột tại biểu tượng tương ứng với loại hộp điều khiển cần chọn cho nó đổi màu.
- Drag chuột ra ngoài biểu mẫu để tạo thành một hộp hình chữ nhật tại vị trí cần đặt biểu tượng.

#### **Cách 2 :**

- Double Click chuột tại biểu tượng tương ứng với loại hộp điều khiển cần chọn. Lúc này hộp điều khiển sẽ tự động được đặt vào giữa biểu mẫu.
- Drag chuột để thay đổi vị trí và kích thước của nó cho đúng với yêu cầu.

#### **Cách 3 :**

- Nhấn giữ phím CTRL đồng thời Click chuột tại biểu tượng tương ứng với loại hộp điều khiển cần chọn.
- Drag chuột ra ngoài biểu mẫu để tạo thành một hộp hình chữ nhật tại vị trí cần đặt biểu tượng.

### **c. Hiệu chỉnh hộp điều khiển**

- Thay đổi vị trí : Drag chuột tại hộp điều khiển trên biểu mẫu.
- Thay đổi kích thước : bấm chuột vào hộp, đưa dấu chuột về góc phải bên dưới rồi Drag chuột.
- Xóa hộp điều khiển : bấm chuột lên hộp rồi gõ phím DELETE.
- Sao chép : Ctrl + C để chép nút vào bộ nhớ.
- Dán : Ctrl + V để dán nút từ bộ nhớ ra màn hình.
- Nhóm các hộp thành một khối : giữ phím Ctrl đồng thời Click chuột lần lượt tại các hộp cần nhóm lại với nhau.

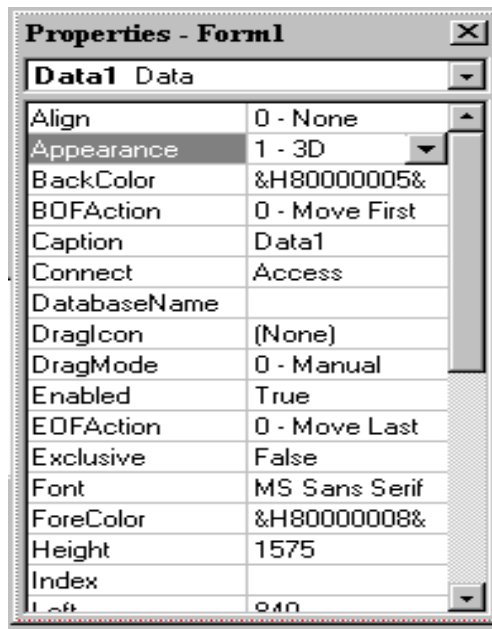
## **2. Thuộc tính**

Thuộc tính là tập hợp các mô tả về đối tượng mà người sử dụng có thể thay đổi được. Ta có thể thay đổi các thuộc tính khi thiết kế biểu mẫu hoặc trong quá trình thực hiện chương trình.

Thông thường, khi thiết kế biểu mẫu ta phải khai báo các thuộc tính có sẵn và ít bị thay đổi. Sau đó khi thực hiện chương trình nếu muốn thay đổi các tham số mô tả thuộc tính thì ta tiếp tục thực hiện các thay đổi đó.

### **a. Khi thiết kế**

- Click chuột vào đối tượng (hộp điều khiển hoặc biểu mẫu) mà ta cần thay đổi thuộc tính.
- Gọi cửa sổ Properties. (nhấn phím F4 hoặc chọn Window - Properties).
- Xuất hiện cửa sổ chứa các thuộc tính như bên. Ta thực hiện thay đổi các thuộc tính theo yêu cầu.
- Cửa sổ thuộc tính :



**b. Khi thực hiện chương trình**

Muốn thay đổi thuộc tính trong quá trình thực hiện chương trình ta viết trong chương trình dòng lệnh:

**[OBJECT.]<PROPERTY> = <NEW VALUE>**

**c. Các loại thuộc tính**

Khi làm việc với các đối tượng ta thường sử dụng các thuộc tính sau :

- **BorderStyle** : qui định dạng xuất hiện của đường viền biểu mẫu. Ta chọn :
  - 0 (none) : không có đường viền. Biểu mẫu không có các nút Minimize, Maximize và hộp trình đơn điều khiển. Không được xê dịch và thay đổi kích thước biểu mẫu.
  - 1 (Fixed Single) : đường viền là gạch đơn. Có các nút thu phóng kích thước. Không thay đổi kích thước.
  - 2 (Sizeable) : cho phép thay đổi kích thước.
  - 3 (Fixed Double) : đường viền nét đôi, cố định vị trí.
- **Caption** : thay đổi nội dung dòng chú thích sẽ hiện lên tại đối tượng.
- **Control Box** : qui định sự có mặt hay không của thanh trình đơn điều khiển trên biểu mẫu. Thuộc tính này không thay đổi được khi chạy chương trình.
- **MinButton** và **MaxButton** : qui định có các nút phóng to (Maximum) hoặc thu nhỏ (Minimum) trên biểu mẫu hay không. Chọn giá trị True hoặc False.
- **Enable** : qui định việc bật (nếu giá trị True) hoặc tắt (giá trị False) sự hoạt động của nút điều khiển hoặc biểu mẫu.
- **Name** : khai báo tên của đối tượng. Tên này giống như tên biến, nó sẽ được sử dụng khi cần làm việc với các đối tượng trong các đoạn mã chương trình.
- **Font** : thay đổi kiểu chữ. Ta có thể sử dụng các thuộc tính sau liên quan đến Font chữ: **FontName** (tên kiểu chữ), **FontSize** (kích thước chữ), **FontBold** (chữ đậm), **FontItalic** (chữ

ngiêng), FontStrikethru (gạch xóa), FontTransparent (nếu True thì có nền), FontUnderline (gạch chân).

- Height : thay đổi chiều cao của đối tượng.
- Width : thay đổi độ rộng của đối tượng
- Left : chuyển dịch đối tượng theo phương ngang.
- Top : chuyển dịch đối tượng theo phương đứng.
- ScaleMode : qui định đơn vị đo lường.
  - 0 : cho biết đơn vị do người dùng ấn định.
  - 1 : trị ngầm định là Twip (1440 Twips = 1 inch).
  - 2 : Point (72 Points = 1 inch).
  - 3 : Pixed (đơn vị đo nhỏ nhất theo độ phân giải màn hình).
  - 4 : Ký tự.
  - 5 : Inch.
  - 6 : mm
  - 7 : cm
- Icon : xác định biểu tượng sẽ được hiển thị vào lúc chạy chương trình khi biểu mẫu này bị thu nhỏ.
- MousePointer : qui định hình dạng của dấu chuột.
- Visible : qui định biểu mẫu được bật hay tắt.
- WindowState : qui định trạng thái của cửa sổ. (0 : bình thường), 1 (thu nhỏ thành biểu tượng) và 2 (phóng to tối đa).
- BackColor : qui định màu nền.
- ForeColor : qui định màu chữ.

Ngoài ra, còn có nhiều thuộc tính khác nhưng ít được dùng. Trong các phần sau nếu gặp sẽ giải thích thêm.

### 3. Thủ tục tình huống

Mỗi một đối tượng (biểu mẫu hoặc hộp điều khiển) đều có thể kèm theo một thủ tục để khi ta kích vào đối tượng này thì thủ tục sẽ được thực hiện. Các thủ tục theo tình huống này tương tự như các Valid trong FOXPRO.

- Cách gọi thủ tục tình huống :

**Cách 1 :** Double Click chuột vào đối tượng tương ứng.

**Cách 2 :** Click chuột vào đối tượng sau đó nhấn phím F7 (hoặc chọn View - Code).

- Cách viết : các chỉ thị được viết vào giữa Sub .... End Sub

**Chú ý :**

- Sau khi đã thiết kế xong biểu mẫu ta lưu trữ ứng dụng vào đĩa bằng cách chọn File - Save rồi đặt tên cho biểu mẫu.

- Muốn thực hiện biểu mẫu ta chọn Run – Start

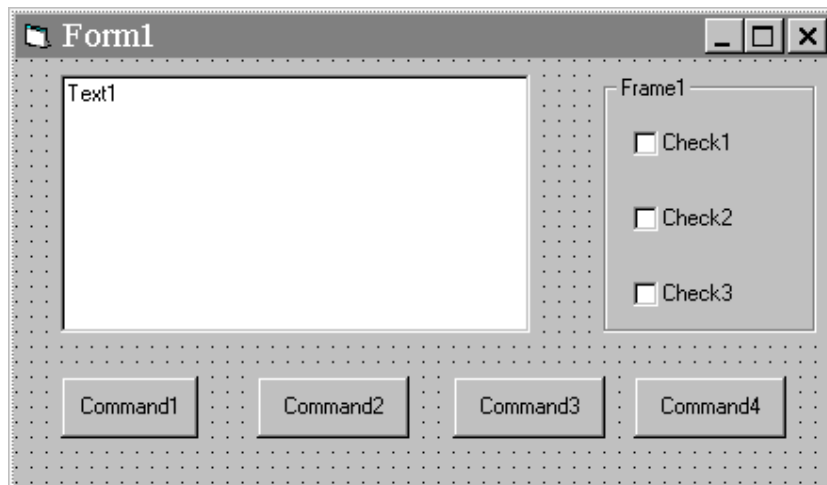
#### IV. Ví dụ

Hãy thiết kế và thực hiện một biểu mẫu để soạn thảo một đoạn văn bản và có thể thực hiện các thao tác để hiệu chỉnh, trang trí cho đoạn văn bản đó.

Để thực hiện công việc trên ta tiến hành theo trình tự sau :

##### 1. Bổ sung hộp điều khiển

- Khởi động Visual Basic.
- Trên cửa sổ Form ta đưa vào lần lượt các hộp điều khiển như sau :
  - 1 hộp Text Box để soạn thảo nội dung văn bản.
  - 3 hộp Check Box để phục vụ việc trang trí.
  - 4 hộp Command Button để phục vụ việc hiệu chỉnh.
- Lúc này ta nhận được mẫu Form như sau :



##### 2. Thay đổi thuộc tính

###### a. Hộp Text

- Click chuột vào hộp Text1, gõ phím F4.
- Đổi các thuộc tính :
  - MultiLine : True.
  - ScrollBar : Both.
  - Name : txt

###### b. Các hộp Command Button

- Lần lượt Click chuột vào các hộp Command1, Command2, Command3 và Command4 và nhấn phím F4.
- Sau đó, ở mỗi lần ta thay đổi :
  - Caption : Cắt, Sao chép, Dán và Xóa (đổi tên tương ứng).

- Name : cmdcut, cmdcopy, cmdpaste và cmddele

**c. Các hộp Check Box**

- Lần lượt Click chuột vào các hộp Check 1, Check 2, Check 3 và nhấn phím F4.
- Sau đó, ở mỗi lần ta thay đổi :
  - Caption : In đậm, In nghiêng và Gạch chân (đổi tên tương ứng).
  - Name : chkbold, chkitalic, chkunder.

**d. Đổi Font**

- Giữ phím Control đồng thời click chuột tại tất cả các đối tượng để tạo nhóm đối tượng.
- Gõ phím F4 gọi cửa sổ Properties.
- Đổi Font sang tên là Vntimes New Roman, Size 10.

**3. Viết các thủ tục tình huống**

**a. Thủ tục của Form : đây là thủ tục chứa các chỉ thị khởi tạo giá trị ban đầu.**

- Bấm Double Click chuột vào nền của Form.
- Gõ vào nội dung chương trình như sau :

```
Private Sub Form_Load()
    txt = ""           'Khởi tạo biến trắng
    txt.FontBold = False      'Không đậm
    txt.FontItalic = False
    txt.FontUnderline = False
    chkbold.Value = 0        'Chưa chọn in đậm
    chkitalic.Value = 0
    chkunder.Value = 0
End Sub
```

- Sau đó vào hộp Object chọn tiếp General để gõ vào chỉ thị khai báo biến nhớ ClipText :

```
Private Sub Text1_Change()
    Dim ClipText As String    'Khởi tạo biến chuỗi tên là ClipText
End Sub
```

**b. Thủ tục của các hộp Command**

- Lần lượt Double Click chuột tại các hộp Cắt, Sao chép, Dán và Xóa và gõ vào các thủ tục tương ứng sau :

```
Private Sub cmdcut_Click()
    ClipText = txt.SelText      'Chép khối vào biến tạm ClipText
    txt.SelText = ""           'Xóa khối khỏi văn bản
    txt.SetFocus                'Tham chiếu đến biến TXT
End Sub
Private Sub cmdcopy_Click()
    ClipText = txt.SelText
    txt.SetFocus
End Sub
Private Sub cmdpaste_Click()
    txt.SelText = ClipText      'Gán khối bởi giá trị biến ClipText
    txt.SetFocus
End Sub
```

```
Private Sub cmdDel_Click()
    txt.Text = ""
    txt.SetFocus
End Sub
```

**c. Thủ tục của các hộp Check Box**

- Lần lượt Double Click chuột tại các hộp In đậm, In nghiêng, Gạch chân và gõ vào các thủ tục sau :

```
Private Sub chkBold_Click()
    txt.FontBold = Not (txt.FontBold)      'Ngược lại với giá trị cũ
    txt.SetFocus
End Sub
Private Sub chkItalic_Click()
    txt.FontItalic = Not (txt.FontItalic)
    txt.SetFocus
End Sub
Private Sub chkUnder_Click()
    txt.FontUnderline = Not (txt.FontUnderline)
    txt.SetFocus
End Sub
```

**4. Ghi và thực hiện chương trình**

- Chọn File - Save để lưu trữ biểu mẫu vào đĩa với tên là VANBAN
- Chọn Run - Start để bắt đầu thực hiện chương trình.

**Chú ý :** nếu muốn lưu trữ và xem nội dung chương trình dưới dạng văn bản thì ta thực hiện như sau :

**a. Lưu trữ**

- Chọn File.
- Chọn File Save As
- Chọn kiểu tập tin là TEXT, đặt tên là Vanban.TXT.

**b. Xem mã lệnh**

- Vào một chương trình soạn thảo văn bản bất kỳ.
- Gõ văn bản dạng TEXT có tên Vanban.TXT.
- Lúc này ta có nội dung chương trình như sau :

```
VERSION 4.00
Begin VB.Form Form1
    Caption           =   "Form1"
    ClientHeight      =   3390
    ClientLeft        =   1140
    ClientTop         =   1665
    ClientWidth       =   6495
    Height            =   3870
    Left              =   1080
    LinkTopic         =   "Form1"
    ScaleHeight       =   3390
    ScaleWidth        =   6495
```



```

Top           = 1245
Width         = 6615
Begin VB.Frame Frame1
    Caption     = "Frame1"
    BeginProperty Font
        name           = "VNtimes new roman"
        charset        = 1
        weight         = 400
        size           = 9.75
        underline      = 0    'False
        italic         = 0    'False
        strikethrough   = 0    'False
    EndProperty
    Height      = 2055
    Left        = 4680
    TabIndex    = 5
    Top         = 120
    Width       = 1695
    Begin VB.CheckBox chkunder
        Caption     = "Gạch chân"
        BeginProperty Font
            name           = "VNtimes new roman"
            charset        = 1
            weight         = 400
            size           = 9.75
            underline      = 0    'False
            italic         = 0    'False
            strikethrough   = 0    'False
        EndProperty
        Height      = 375
        Left        = 240
        TabIndex    = 8
        Top         = 1560
        Width       = 1215
    End
    Begin VB.CheckBox chkitalic
        Caption     = "In nghiêng"
        BeginProperty Font
            name           = "VNtimes new roman"
            charset        = 1
            weight         = 400
            size           = 9.75
            underline      = 0    'False
            italic         = 0    'False
            strikethrough   = 0    'False
        EndProperty
        Height      = 375
        Left        = 240
        TabIndex    = 7
        Top         = 960
        Width       = 1215
    End
    Begin VB.CheckBox chkbold

```

```

Caption      = "In đậm"
BeginProperty Font
    name      = "VNtimes new roman"
    charset   = 1
    weight    = 400
    size      = 9.75
    underline  = 0 'False
    italic     = 0 'False
    strikethrough = 0 'False
EndProperty
Height       = 375
Left        = 240
TabIndex    = 6
Top         = 360
Width       = 1215
End
End
Begin VB.CommandButton cmdddel
Caption      = "Xóa"
BeginProperty Font
    name      = "VNtimes new roman"
    charset   = 1
    weight    = 400
    size      = 9.75
    underline  = 0 'False
    italic     = 0 'False
    strikethrough = 0 'False
EndProperty
Height       = 495
Left        = 4920
TabIndex    = 4
Top         = 2520
Width       = 1215
End
Begin VB.CommandButton cmdpaste
Caption      = "Dán"
BeginProperty Font
    name      = "VNtimes new roman"
    charset   = 1
    weight    = 400
    size      = 9.75
    underline  = 0 'False
    italic     = 0 'False
    strikethrough = 0 'False
EndProperty
Height       = 495
Left        = 3480
TabIndex    = 3
Top         = 2520
Width       = 1215
End
Begin VB.CommandButton cmdcopy
Caption      = "Sao chép"

```

```

BeginProperty Font
    name            =    "VNtimes new roman"
    charset         =    1
    weight          =    400
    size            =    9.75
    underline       =    0    'False
    italic          =    0    'False
    strikethrough   =    0    'False
EndProperty
Height            =    495
Left              =    1920
TabIndex         =    2
Top              =    2520
Width            =    1215
End
Begin VB.CommandButton cmdcut
Caption          =    "Cắt"
BeginProperty Font
    name            =    "VNtimes new roman"
    charset         =    1
    weight          =    400
    size            =    9.75
    underline       =    0    'False
    italic          =    0    'False
    strikethrough   =    0    'False
EndProperty
Height          =    495
Left            =    360
TabIndex       =    1
Top            =    2520
Width         =    1095
End
Begin VB.TextBox txt
BeginProperty Font
    name            =    "VNtimes new roman"
    charset         =    1
    weight          =    400
    size            =    9.75
    underline       =    0    'False
    italic          =    0    'False
    strikethrough   =    0    'False
EndProperty
Height          =    2055
Left            =    360
MultiLine      =    -1    'True
ScrollBars     =    3    'Both
TabIndex       =    0
Text           =    "THU1.frx":0000
Top            =    120
Width          =    3735
End
End
Attribute VB_Name = "Form1"

```

```

Attribute VB_Creatable = False
Attribute VB_Exposed = False
Private Sub chkbold_Click()
txt.FontBold = Not (txt.FontBold)
txt.SetFocus
End Sub

Private Sub chkitalic_Click()
txt.FontItalic = Not (txt.FontItalic)
txt.SetFocus
End Sub

Private Sub chkunder_Click()
txt.FontUnderline = Not (txt.FontUnderline)
txt.SetFocus
End Sub

Private Sub cmdcopy_Click()
ClipText = txt.SelText
txt.SetFocus
End Sub

Private Sub cmdcut_Click()
ClipText = txt.SelText 'Chép khối lên bộ nhớ
txt.SelText = ""
txt.SetFocus
End Sub

Private Sub cmddel_Click()
txt.Text = ""
txt.SetFocus
End Sub

Private Sub cmdpaste_Click()
txt.SelText = ClipText
txt.SetFocus
End Sub
Private Sub Form_Load()
Dim ClipText As String
txt = ""
txt.FontBold = False
txt.FontItalic = False
txt.FontUnderline = False
chkbold.Value = 0
chkitalic.Value = 0
chkunder.Value = 0
End Sub
Private Sub Text1_Change()
Dim ClipText As String
End Sub

```

## V. Biến nhớ

### 1. Khái niệm

Trong Visual Basic ta có thể lưu trữ các giá trị để phục vụ cho quá trình xử lý dữ liệu dưới bốn hình thức là : biến, hằng, mảng và bản ghi.

Biến và hằng tại mỗi thời điểm chỉ lưu trữ được một giá trị còn mảng và bản ghi thì lưu trữ được nhiều giá trị cùng lúc.

Tên các đại lượng do người sử dụng tự qui định nhưng phải thỏa mãn :

- Không dài quá 40 ký tự.
- Ký tự đầu tiên là chữ cái, các ký tự đi sau có thể là số, dấu \_ (gạch dưới).
- Ký tự cuối cùng có thể là một trong các hậu tố cho biết kiểu dữ liệu như : %, &, !, #, @ và #.
- Tên biến không được trùng với các từ dành riêng (Reserved word).
- Visual Basic không phân biệt ký tự chữ in với chữ thường.

### 2. Khai báo biến

Ta có thể định nghĩa biến bằng một trong các cách sau :

#### a. Cách 1

**DIM <Tên biến> AS <Tên kiểu dữ liệu>**

Trong đó tên kiểu dữ liệu được chọn từ một trong các kiểu sau :

- Integer (2 byte): là số nguyên bình thường thuộc [-32768, 32767].
- Long (4 byte): số nguyên dài thuộc [-2147483648, 2147483647].
- Single (4 byte): số thực độ chính xác đơn thuộc [-3.4E+38, 3.4E+38]. Giữ lại 7 chữ số sau dấu thập phân.
- Double (8 byte): số thực độ chính xác kép thuộc [-1.8E+308, 1.8E+308]. Giữ lại 16 chữ số sau dấu thập phân
- Currency (8 byte) : lưu trữ các giá trị là tiền tệ. Chứa được tối đa 15 chữ số bên trái và 4 chữ số bên phải dấu thập phân.
- String :chứa chuỗi ký tự có chiều dài thay đổi từ 0 đến 65535 ký tự.
- String\*num : chứa chuỗi ký tự có chiều dài định trước. Khi khai báo phải ấn định trước độ dài từ 0 đến 32767 ký tự.
- Varian : lưu trữ đồng thời hai thông tin là : giá trị và kiểu dữ liệu.
  - Phạm vi : các chỉ thị DIM có thể xuất hiện ở cấp đơn thể, cấp biểu mẫu hoặc cấp thủ tục. Nếu ta khai báo DIM ở cấp nào thì các biến chỉ có hiệu lực trong cấp đó.
  - Ví dụ : DIM sotien AS long  
DIM luong AS Currency  
DIM hoten AS String

## **b. Cách 2**

**DIM <Tên biến><ký tự hậu tố>**

Ta có thể không cần dùng AS <kiểu dữ liệu> mà ghi trực tiếp ký hiệu hậu tố khai báo kiểu dữ liệu vào sau tên biến để định kiểu.

Ta có các ký tự hậu tố (Suffix Character) như sau :

- % : Integer.
- & : Long.
- ! : Single.
- # : Double.
- @ : Currency.
- \$ : String.

**Ví dụ :**

DIM hoten\$

DIM sotien@

DIM luong%

## **c. Khai báo biến toàn cục**

**GLOBAL <Tên biến> AS <Tên kiểu dữ liệu>**

Trong trường hợp biến sẽ có hiệu lực trên tất cả các đơn thể của chương trình. Lệnh này chỉ được sử dụng ở phần declarations của đơn thể.

Ví dụ : GLOBAL hoten AS String

Lúc này biến hoten sẽ có hiệu lực trên toàn chương trình.

## **d. Khai báo nhiều biến**

**DefType <miền ký tự>**

Trong Visual Basic nếu một biến được sử dụng mà không khai báo thì Visual Basic ngầm hiểu là biến Varian. Với chỉ thị DefType ta có thể chuyển biến Varian thành một kiểu dữ liệu khác.

- Visual Basic chấp nhận các chỉ thị DefType sau :
  - DefInt : Integer.
  - DefLng : Long.
  - DefSng : Single.
  - DefDbl : Double.
  - DefCur : Currency.
  - DefStr : String.
- DefVar : Varian.
- Miền ký tự : cho biết khoảng ký tự. Từ đây về sau những biến nào có ký tự bắt đầu thuộc miền ký tự trên sẽ có kiểu dữ liệu như trong DefType qui định.

**Ví dụ :** DefInt S

Từ đây về sau các biến như : Sotien, Soluong, S... đều có kiểu dữ liệu là Integer vì có ký tự mở đầu là S

Tuy nhiên chỉ thị này không ảnh hưởng đến các biến được khai báo bởi chỉ thị DIM hoặc mang các hậu tố qui định kiểu dữ liệu.

### 3. Khai báo hằng

Hằng (Constant) là một đại lượng có giá trị không thay đổi trong suốt thời gian thực hiện chương trình.

Cách khai báo :

**CONST <Tên hằng> = <Biểu thức>, <Tên hằng 2> = <Biểu thức 2>, ...**

Tên hằng cũng theo qui ước giống như tên biến. Ta có thể gắn ký tự hậu tố để định kiểu (% , !, &, #, \$, @) cho hằng. Tuy nhiên, sau này khi dùng tên hằng trong chương trình thì không được viết hậu tố này.

**Ví dụ :**

CONST diemgioi% = 7

Sau này trong chương trình ta chỉ viết :

IF dtb% >= diemgioi THEN MsgBox "Sinh viên này xếp loại Giỏi"

Phạm vi hoạt động của hằng giống như của biến. Nếu muốn khai báo hằng có tác dụng toàn cục thì viết :

GLOBAL CONST <Tên hằng> = <Biểu thức>

### 4. Mảng

Mảng (Array) là đại lượng có thể lưu trữ được nhiều giá trị khác nhau tại cùng một thời điểm thông qua các phần tử của nó.

**a. Khai báo mảng**

**DIM <Tên mảng>(phần tử) AS <Kiểu dữ liệu>**

hoặc **DIM <Tên mảng><Hậu tố kiểu>(phần tử)**

- Tên mảng do người sử dụng tự qui định. Giống như cách khai báo tên biến.

- Phần tử : khai báo số lượng các phần tử trong mảng. Có nhiều cách :

Số lượng tối đa. Trong trường hợp này phần tử bắt đầu là không.

**Ví dụ :** DIM a(10) AS Integer hoặc DIM a%(10)

Lúc này sẽ có các phần tử là a(0), a(1), ..., a(10) và mỗi phần tử chứa một số nguyên.

Phần tử bắt đầu đến phần tử cuối. Qui định rõ các các phần tử đầu đến cuối.

**Ví dụ :** DIM A(5 TO 10) AS Single hoặc DIM A!(5 TO 10)

Lúc này có các phần tử là a(5), a(6), ..., a(10) và mỗi phần tử là số thực độ chính xác đơn..

Mảng nhiều chiều. Giữa các chiều ngăn cách bởi dấu , (phẩy).

**Ví dụ :** DIM a(10,20) AS Integer hoặc DIM a%(10,20)

Lúc này sẽ có các phần tử là  $a(0,0)$ ,  $a(0,1)$ , ....  $a(10,20)$  và mỗi phần tử chứa một số nguyên.

`DIM A(1 TO 5, 1 TO 7) AS Single`

Lúc này sẽ có các phần tử là  $a(1,1)$ ,  $a(1,2)$ , ....  $a(5,7)$  và mỗi phần tử chứa một số thực.

Phạm vi hoạt động của biến mảng cũng giống như các biến bình thường khác.

- Khai báo bằng GLOBAL: lúc này biến sẽ có tác dụng trên toàn chương trình. Khai báo này phải đặt trong phần khai báo của đơn thể chứ không đặt trong thủ tục hoặc biểu mẫu.

#### **b. Sử dụng mảng**

Biến mảng được sử dụng trong chương trình giống như các biến thông thường khác, tuy nhiên phải chỉ rõ số hiệu phần tử của nó.

Ví dụ : khai báo biến mảng tháng và lưu trữ số thứ tự của tháng trong năm. Sau đó in các tên tháng ra màn hình.

```
DIM thang(1 TO 12) AS Integer
FOR I = 1 TO 12
    Thang(I) = I
NEXT I
FOR I = 1 TO 12
    Print "Thang :" + Str(thang(I))
NEXT I
```

### **5. Khai báo bản ghi**

Bản ghi là kiểu dữ liệu đặc biệt bao gồm nhiều giá trị thuộc nhiều kiểu dữ liệu khác nhau. Biến bản ghi được sử dụng nhiều để giải quyết các bài toán trong quản lý số liệu.

#### **a. Khai báo**

Khai báo bản ghi bắt buộc phải được đặt ở phân đoạn Declaration của đơn thể chương trình mà không thể đặt ở cấp biểu mẫu hoặc thủ tục. Biến bản ghi đương nhiên phải là biến toàn cục.

Cách khai báo :

```
TYPE      <Tên bản ghi>
    <Tên trường 1>    AS  <Tên kiểu dữ liệu 1>
    <Tên trường 2>    AS  <Tên kiểu dữ liệu 2>
    .....
    <Tên trường n>    AS  <Tên kiểu dữ liệu n>
END TYPE
DIM <Tên biến> AS <Tên bản ghi>
```

Tên bản ghi do người sử dụng tự qui định theo qui tắc của tên biến.

#### **b. Sử dụng biến bản ghi**

Các biến bản ghi được sử dụng như các biến bình thường khác nhưng phải chỉ rõ tên trường ở phía sau và ngăn cách với tên biến bởi dấu . (chấm).

Cách viết : <Tên biến bản ghi> . <Tên trường>

### **6. Biến đổi (convert) từ loại dữ liệu này qua loại dữ liệu khác**

Nhiều lúc ta cần phải convert data type của một variable từ loại này qua loại khác, VB6 cho ta một số các hàm dưới đây. Xin lưu rằng khi gọi các hàm này, nếu chúng ta đưa một data value bất hợp lệ thì có thể bị lỗi.



Tên hàm	Chức năng
CBool ()	Đổi parameter ra True hay False. Nếu Integer khác 0 thì được đổi thành True
CByte ()	Đổi parameter ra một con số từ 0 đến 255 nếu có thể được, nếu không thì là 0.
CDate ()	Đổi parameter ra Date
CDBl ()	Đổi parameter ra Double precision floating point number
CInt ()	Đổi parameter ra Integer
CSng ()	Đổi parameter ra Single precision floating point number
CStr ()	Đổi parameter ra String

Ngoài các hàm nói trên chúng ta cũng có thể dùng hàm **Val** để chuyển đổi một String ra Number. Lưu ý là khi hàm đổi chuỗi sang số gặp một ký tự nào đó không phải là chữ số hay dấu chấm thập phân thì nó không xử lý tiếp nữa. Do đó nếu Input String là "\$25.50" thì giá trị số sẽ là 0 vì \$ không phải là một chữ số. Nếu Input String là "62.4B" thì giá trị trả về là 62.4.

CDBl là hàm dùng để convert một String ra số an toàn nhất. Input String có thể chứa các dấu , và . (thí dụ: 1,234,567.89) tùy theo nơi chúng ta ở trên thế giới (thí dụ như Âu Châu hay Mỹ). CSng cũng làm việc giống như CDBl nhưng nếu con số lớn hơn 1 triệu nó có thể bị bug.

Lỗi hay gặp của CSng là nếu Input String không có gì cả (tức là InputString="") thì hàm CSng cho chúng ta "Type Mismatch Error". Do đó để khắc phục khuyết điểm này chúng ta có thể viết cho mình một hàm tạm đặt tên là CSingle để dùng thế cho CSng :

```
Function CSingle(strNumber) As Single
    If Trim(strNumber) = "" Then
        CSingle = 0#
    Else
        CSingle = CSng(strNumber)
    End If
End Function
```

## CHƯƠNG 4. VIẾT MÃ CHƯƠNG TRÌNH

### I. Các cấu trúc điều khiển

Tương tự như các ngôn ngữ lập trình khác, trong Visual Basic ta có thể sử dụng các cấu trúc điều khiển trong chương trình để có thể chọn lựa công việc thực hiện hoặc tự động lặp lại nhóm chỉ thị nhiều lần.

#### 1. Cấu trúc chọn

##### a. Cấu trúc

**IF <Điều kiện> THEN <Chỉ thị>**

Khi gặp cấu trúc này nếu điều kiện có giá trị True thì thực hiện chỉ thị nếu điều kiện có giá trị False thì bỏ qua chỉ thị đó.

**Ví dụ :**

```
IF dtb > 5 THEN Print "Bạn đủ điểm"
```

Trong trường hợp này chỉ có duy nhất một chỉ thị.

##### b. Cấu trúc

**IF <Điều kiện> THEN <Chỉ thị 1> ELSE <Chỉ thị 2>**

Khi gặp cấu trúc này nếu điều kiện có giá trị True thì thực hiện chỉ thị 1 nếu điều kiện có giá trị False thì thực hiện chỉ thị 2.

**Ví dụ :**

```
IF dtb > 5 THEN Print "Bạn đủ điểm" ELSE Print "Bạn thiếu điểm"
```

**Chú ý :**

- Nếu muốn sau THEN hoặc ELSE có nhiều chỉ thị cần thực hiện thì phải viết xuống dòng và cuối cấu trúc này phải có END IF.

**Cách viết :**

```
IF <Điều kiện> THEN
    <Chỉ thị 1>
    .....
    <Chỉ thị n>
ELSE
    <Chỉ thị 1'>
    .....
    <Chỉ thị n'>
END IF
```

##### c. Cấu trúc

**SELECT CASE <Biểu thức>**

```
Select Case <biểu thức>
Case <Liệt kê biểu thức 1>
    <Khối chỉ thị 1>
```

```

Case <Liệt kê biểu thức 2>
    <Khối chỉ thị 2>
    .....
[Case Else
    <Khối chỉ thị n>]
End Select

```

Trong đó :

- Biểu thức : là một thức chuỗi hoặc số. Nếu giá trị của biểu thức ở đây trùng với giá trị của các biểu thức được liệt kê nào bên dưới thì khối chỉ thị tương ứng được thực hiện.
- Liệt kê biểu thức I : là biểu thức sẽ được đem so sánh với biểu thức đầu. Trong phần này biểu thức liệt kê có thể được viết dưới các dạng sau :

- Biểu thức : số hoặc chữ.
- Biểu thức 1 TO Biểu thức 2 : chỉ ra đoạn giá trị nằm giữa biểu thức 1 và 2.
- IS <phép so sánh> <biểu thức> : chỉ ra phép so sánh và giá trị so sánh.

- Khối chỉ thị I : là các chỉ thị cần thực hiện trong trường hợp giá trị của biểu thức thứ I trùng với giá trị của biểu thức đầu. Ở đây có thể gồm nhiều chỉ thị được viết trên nhiều dòng.

Ví dụ : viết chương trình nhập vào tuổi một người và cho biết người đó thuộc lứa tuổi nào.

```

Sub Form_Click()
    Dim Cauhoi, Tuoi 'Khai báo biến Cauhoi và Tuoi
    Cauhoi = "Bạn bao nhiêu tuổi :"
    Tuoi = InputBox(Cauhoi) ' Nhập tuổi vào biến tuoi
    Select Case Tuoi
        Case 1 TO 12
            MsgBox "Bạn ở tuổi Nhi đồng" 'In ra dòng thông báo
        Case 13 TO 18
            MsgBox "Bạn ở tuổi Thiếu niên"
        Case 18 TO 25
            MsgBox "Bạn ở tuổi Thanh niên"
        Case 25 TO 60
            MsgBox "Bạn đã là Người lớn"
        Case IS > 60
            MsgBox "Bạn ở tuổi Về hưu"
        Case Else
            MsgBox "Bạn không phải con người"
    End Select
End Sub

```

## 2. Cấu trúc lặp

### a. Cấu trúc

```

FOR <Biến đếm> = <Giá trị đầu> TO <Giá trị cuối> [STEP n]
    [Khối chỉ thị 1]
[Exit For]
    [Khối chỉ thị 2]
NEXT <biến đếm>

```

- Biến đếm : là tên biến dùng để kiểm tra số lần lặp.
- Giá trị đầu : là giá trị khởi gán lần đầu tiên cho biến đếm khi thực hiện vòng lặp.
- Giá trị cuối : là giá trị cuối cùng của biến đếm. Khi biến đếm đạt đến giá trị này thì vòng lặp thực hiện lần cuối và dừng quá trình lặp.
- STEP n : chỉ định bước nhảy n để thực hiện thay đổi giá trị của biến đếm sau mỗi lần lặp.
- Khối chỉ thị : liệt kê các chỉ thị cần thực hiện trong mỗi lần lặp.
- Exit For : nếu trong vòng lặp mà gặp chỉ thị này thì sẽ ngưng vòng lặp.

Vòng lặp trên cho phép tự động thực hiện các chỉ thị với số lần lặp xác định trước.

Ví dụ : viết đoạn lệnh in ra các số nguyên từ 1 đến 10.

```
FOR so! = 1 TO 10                                'biến số là Single
  Print so!
NEXT so!
```

Ví dụ : viết đoạn lệnh in ra các số với bước nhảy 0.25 và từ 0 đến 10.

```
FOR so! = 0 TO 10 STEP 0.25                      'biến số là Single
  Print so!
NEXT so!
```

## b. Cấu trúc

**WHILE <Điều kiện>**

**[Khối chỉ thị]**

**Wend**

- Điều kiện : qui định điều kiện để thực hiện vòng lặp. Nếu điều kiện có giá trị True thì thực hiện khối chỉ thị, gặp Wend sẽ quay trở lại kiểm tra điều kiện. Quá trình trên kết thúc khi điều kiện có giá trị False.
- Khối chỉ thị : các chỉ thị cần thực hiện trong vòng lặp.

Ví dụ : viết đoạn lệnh in ra các số nguyên từ 1 đến 10.

```
So! = 1
While so! <= 10
  Print so!
  So! = so! + 1
Wend
```

## c. Cấu trúc

**DO [WHILE | UNTIL <Điều kiện>]**

**[Khối chỉ thị]**

**[Exit Do]**

**[Khối chỉ thị]**

**LOOP [WHILE | UNTIL <Điều kiện>]**

- Điều kiện : qui định điều kiện để thực hiện vòng lặp.

- Nếu dùng WHILE thì điều kiện có giá trị True thì thực hiện khối chỉ thị, nếu False kết thúc vòng lặp.
- Nếu dùng UNTIL thì điều kiện có giá trị False thì thực hiện khối chỉ thị, nếu True kết thúc vòng lặp.

Ta có thể đặt điều kiện ở đầu hoặc cuối vòng lặp đều được.

- Khối chỉ thị : các chỉ thị cần thực hiện trong vòng lặp.
- Exit Do : cho phép dừng vòng lặp mà không cần qua kiểm tra điều kiện.

Ví dụ : viết đoạn lệnh in ra các số nguyên từ 1 đến 10.

```
So! = 1
Do While so! <= 10
    Print so!
    So! = so! + 1
Loop
```

Hoặc :

```
So! = 1
Do
    Print so!
    So! = so! + 1
Loop While so! <= 10
```

Hoặc :

```
So! = 1
Do
    Print so!
    So! = so! + 1
Loop Until so! > 10
```

### 3. Nhãn

Trong Visual Basic ta có thể chuyển đến thực hiện ở một đoạn chương trình hoặc một dòng lệnh mới bằng cách dùng nhãn hoặc số thứ tự dòng lệnh.

#### a. Khái niệm

Là một đoạn chỉ thị lệnh bất kỳ trong chương trình được gán một tên xác định. Khi cần thực hiện đoạn chỉ thị này ta chỉ việc nhảy về nhãn đó.

Mỗi nhãn được dùng trong biểu mẫu hoặc đơn thể phải là duy nhất. Không thể sử dụng hai nhãn trùng tên trong một biểu mẫu, thủ tục, hộp điều khiển...

#### b. Cách viết

**Tên\_nhãn: <Nhóm chỉ thị>**

#### c. Cách gọi sử dụng

- Cách 1 : sử dụng lệnh **GOTO <tên\_nhãn>**
- Cách 2 : sử dụng lệnh **ON <biểu thức số> GOTO <liệt kê các tên nhãn>**

Trong trường hợp này biểu thức số có giá trị từ 1 đến 255 và tên nhãn có số thứ tự tương ứng với biểu thức số sẽ được thực hiện.

**Ví dụ :** ON stt GOTO nhan1, nhan2, nhan3

Nếu stt có giá trị 1 thì nhan1 được thực hiện, stt là 2 thì nhan2 thực hiện và stt là 3 thì nhan3 được thực hiện.

**Ví dụ :**

```
Sub Form_Click()  
Print "Giáo trình"  
GOTO Nhan1  
Print "Không in"  
Nhan1:  
Print "Lập trình trực quan"  
End Sub
```

Lúc này kết quả trên màn hình ta nhận được :

Giáo trình

Lập trình trực quan

Còn dòng lệnh Print "Không in" sẽ không thực hiện.

#### 4. Số thứ tự dòng lệnh

Là phương pháp đánh số ở trước mỗi dòng lệnh và khi cần ta có nhảy đến vị trí này bất cứ lúc nào.

Mỗi số được dùng trong biểu mẫu hoặc đơn thể phải là duy nhất. Không thể sử dụng hai số trùng giá trị để đánh số dòng lệnh trong một biểu mẫu, thủ tục, hộp điều khiển...

Các số dùng đánh số dòng lệnh là tùy ý không bắt buộc phải đánh số theo thứ tự tăng hay giảm dần mà ngẫu nhiên, không bắt buộc phải đánh số tất cả các chỉ thị lệnh mà thích đánh số vào dòng lệnh nào cũng được.

Cách gọi : GOTO <giá trị số>

Khi thực hiện lệnh này Visual Basic sẽ chuyển đến dòng lệnh được đánh số tương ứng.

**Ví dụ :**

```
100 MsgBox "Dòng lệnh mang số 100"  
101 MsgBox "Dòng lệnh mang số 101"  
57 MsgBox "Dòng lệnh mang số 57"  
GOTO 101
```

## II. Method

Method là các chương trình được xây dựng sẵn để phục vụ cho việc thực hiện các thao tác thường gặp. Method có tác dụng gần giống như lệnh, thủ tục hoặc hàm được xây dựng sẵn trong các ngôn ngữ lập trình có cấu trúc. Thông thường Method chỉ tác dụng lên một lớp các đối tượng.

Sau đây ta sẽ xét một số Method thường được sử dụng.

### 1. Circle Method

Cú pháp : [Object].Circle [Step] (X,Y), Radius [, [Color], [Start], [End], [Aspect]]

- Object : tên của biểu mẫu hoặc khung hình mà ta cần vẽ hình tròn trên đó.
- Step : cho biết đây là tọa độ tương đối so với vị trí hiện hành do hai thuộc tính CurrentX và CurrentY cung cấp.

- X, Y : chỉ định tọa độ tâm của hình tròn, ellipse hoặc cung tròn.
- Radius : chỉ định bán kính.
- Color : chỉ định màu sắc. Màu tương ứng với giá trị là một số nguyên.
- Start, End : trị số tính theo Radian, cho biết điểm xuất phát và điểm kết thúc khi vẽ một cung tròn hoặc Ellipse.
- Aspect : trị số cho biết góc xoay mặt phẳng chứa hình tròn để tạo ra hình Ellipse.

Tác dụng : cho phép tạo ra một hình tròn, cung tròn hoặc hình Ellipse theo yêu cầu người sử dụng.

Ví dụ 1: Form.Circle (1000, 2000), 500

Vẽ hình tròn có tâm là điểm (1000,2000) và bán kính là 500. (các đơn vị tính theo Fixed).

Ví dụ 2 : vẽ một dãy các hình tròn đồng tâm với màu sắc tùy ý.

```
Private Sub Hinhtron_Click()
    Dim CX, CY, Radius ' Declare variable.
    ScaleMode = 3 ' Set scale to pixels.
    CX = ScaleWidth / 2 ' Set X position.
    CY = ScaleHeight / 2 ' Set Y position.
    If CX > CY Then Limit = CY Else Limit = CX
    For Radius = 0 To Limit ' Set radius.
        Circle (CX, CY), Radius, RGB(Rnd * 255, Rnd * 255, Rnd *
255)
    Next Radius
End Sub
```

## 2. Line Method

Cú pháp : **[Object].Line [Step] (X1, Y1) - [Step] (X2, Y2)[,Color][,BF]**

- Object : tên của biểu mẫu hoặc khung hình mà ta cần vẽ đường thẳng trên đó.
- X1, Y1 : chỉ định tọa độ điểm xuất phát.
- X2, Y2 : chỉ định tọa độ điểm kết thúc.
- Color : chỉ định màu sắc. Màu tương ứng với giá trị là một số nguyên.
- Step : cho biết đây là tọa độ tương đối so với vị trí hiện hành do hai thuộc tính CurrentX và CurrentY cung cấp.
- B (Box) : vẽ một khung hình chữ nhật. Lúc này điểm xuất phát và điểm kết thúc là hai góc hình chữ nhật.
- F (Fill) : khung hình chữ nhật sẽ được tô màu.

Tác dụng : cho phép tạo ra một đoạn thẳng hoặc khung hình chữ nhật theo yêu cầu người sử dụng.

Ví dụ : vẽ các hình và các đường thẳng với nhiều màu sắc và hình dạng khác nhau.

```
Private Sub duongthang_Click()
    Dim CX, CY, F, F1, F2, I ' Declare variables
    ScaleMode = 3 ' Set ScaleMode to pixels.
    CX = ScaleWidth / 2 ' Get horizontal center.
    CY = ScaleHeight / 2 ' Get vertical center.
```

```

        DrawWidth = 8          ' Set DrawWidth.
    For I = 50 To 0 Step -2
        F = I / 50             ' Perform interim
        F1 = 1 - F: F2 = 1 + F ' calculations.
        ForeColor = QBColor(I Mod 15) ' Set foreground color.
        Line (CX * F1, CY * F1)-(CX * F2, CY * F2), , BF
    Next I
    DoEvents                   ' Yield for other processing.
    If CY > CX Then             ' Set DrawWidth.
        DrawWidth = ScaleWidth / 25
    Else
        DrawWidth = ScaleHeight / 25
    End If
    For I = 0 To 50 Step 2      ' Set up loop.
        F = I / 50             ' Perform interim
        F1 = 1 - F: F2 = 1 + F ' calculations.
        Line (CX * F1, CY)-(CX, CY * F1) ' Draw upper-left.
        Line -(CX * F2, CY) ' Draw upper-right.
        Line -(CX, CY * F2) ' Draw lower-right.
        Line -(CX * F1, CY) ' Draw lower-left.
        ForeColor = QBColor(I Mod 15) ' Change color each time.
    Next I
    DoEvents                   ' Yield for other processing.
End Sub

```

### 3. Cls Method

Cú pháp : **[object.]Cls**

Tác dụng : xóa màn hình của Form.

Ví dụ :

```

Private Sub Xoa_Click()
    Dim Msg ' Declare variable.
    AutoRedraw = -1 ' Turn on AutoRedraw.
    ForeColor = QBColor(15) ' Set foreground to white.
    BackColor = QBColor(1) ' Set background to blue.
    FillStyle = 7 ' Set diagonal crosshatch.
    Line (0, 0)-(ScaleWidth, ScaleHeight), , B ' Put box on form.
    Msg = "This is information printed on the form background."
    CurrentX = ScaleWidth / 2 - TextWidth(Msg) / 2 ' Set X position.
    CurrentY = 2 * TextHeight(Msg) ' Set Y position.
    Print Msg ' Print message to form.
    Msg = "Choose OK to clear the information and background "
    Msg = Msg & "pattern just displayed on the form."
    MsgBox Msg ' Display message.
    Cls ' Clear form background.
End Sub

```

### 4. Hide Method

Cú pháp : **[Object.]Hide**

Tác dụng : che cửa sổ Form.

Ví dụ : che và làm xuất hiện lại cửa sổ Form đang làm việc..



```
Private Sub Chehien_Click()
Dim Msg ' Declare variable.
Hide    ' Hide form.
Msg = "Choose OK to make the form reappear."
MsgBox Msg ' Display message.
Show    ' Show form again.
End Sub
```

## 5. Show Method

Cú pháp : **[Object.]Show**

Tác dụng : làm xuất hiện cửa sổ Form.

```
Private Sub Chehien_Click()
Dim Msg ' Declare variable.
Hide    ' Hide form.
Msg = "Choose OK to make the form reappear."
MsgBox Msg ' Display message.
Show    ' Show form again.
End Sub
```

## 6. Item Method

Cú pháp : **[Object.]Item(Index)**

Tác dụng : sắp xếp lại các thành viên trong Collection theo thứ tự của khóa chỉ định trong Index.

Ví dụ :

```
Dim SmithBillBD As Object
Dim SmithAdamBD As Object
Set SmithBillBD = Birthdays.Item("SmithBill")
Set SmithAdamBD = Birthdays("SmithAdam")
```

## 7. Move Method

Cú pháp : **[Object.]Move Left [, Top][, Width][, Height]**

- Object: tên Object cần chuyển dịch.
- Left : qui định giá trị cần dịch chuyển sang bên trái.
- Top : qui định dịch chuyển lên phía trên.
- Width : qui định độ rộng mới của đối tượng.
- Height : qui định độ cao mới của đối tượng.

Tác dụng : cho phép di chuyển và điều chỉnh kích thước của đối tượng.

Ví dụ :

```
Private Sub dichuyen_Click()
Dim Inch, Msg ' Declare variables.
Msg = "Choose OK to resize and move this form by "
Msg = Msg & "changing the value of properties."
MsgBox Msg ' Display message.
Inch = 1440 ' Set inch in twips.
Width = 4 * Inch ' Set width.
Height = 2 * Inch ' Set height.
```

```

Left = 0      ' Set left to origin.
Top = 0      ' Set top to origin.
Msg = "Now choose OK to resize and move this form "
Msg = Msg & "using the Move method."
MsgBox Msg    ' Display message.
Move Screen.Width-2*Inch, Screen.Height-Inch, 2*Inch, Inch
End Sub

```

## 8. Point Method

Cú pháp : **[Object.]Point (X, Y)**

- X : Hoành độ của điểm cần vẽ.
- Y : Tung độ của điểm cần vẽ.

Tác dụng : trả về một điểm có toạ độ xác định.

Ví dụ : tô màu bằng các dấu chấm.

```

Private Sub vediem_Click()
    Dim LeftColor, MidColor, Msg, RightColor 'Declare variables.
    AutoRedraw = -1 ' Turn on AutoRedraw.
    Height = 3 * 1440 ' Set height to 3 inches.
    Width = 5 * 1440 ' Set width to 5 inches.
    BackColor = QBColor(1) ' Set background to blue.
    ForeColor = QBColor(4) ' Set foreground to red.
    Line (0, 0)-(Width / 3, Height), , BF ' Red box.
    ForeColor = QBColor(15) ' Set foreground to white.
    Line (Width / 3, 0)-((Width / 3) * 2, Height), , BF
    LeftColor = Point(0, 0) ' Find color of left box,
    MidColor = Point(Width / 2, Height / 2) ' middle box, and
    RightColor = Point(Width, Height) ' right box.
    Msg = "The color number for the red box on the left side of "
    Msg = Msg & "the form is " & LeftColor & ". The "
    Msg = Msg & "color of the white box in the center is "
    Msg = Msg & MidColor & ". The color of the blue "
    Msg = Msg & "box on the right is " & RightColor & "."
    MsgBox Msg ' Display message.
End Sub

```

## 9. Print Method

Cú pháp : **[Object.]Print OutputList**

Tác dụng : cho phép in giá trị các biểu thức trong OutputList ra đối tượng. OutputList là một danh sách các biểu thức cần in. Object là tên đối tượng mà ta cần in lên đó.

Nếu muốn in máy in thì tên Object là Printer. Ví dụ :

```

Private Sub Command1_Click()
    Dim MyVar
    MyVar = "Chúc các bạn lập trình thật tốt với Visual Basic."
    Print MyVar
End Sub

```

## 10. PrintForm Method

Cú pháp : **[Object.]PrintForm**

Tác dụng : cho phép in tất cả các hình ảnh của biểu mẫu ra giấy. Nếu không chỉ rõ tên Form thì Form đang làm việc sẽ được in. Object ở đây là tên Form cần in.

Ví dụ

```
Private Sub Command1_Click()  
    Dim Msg ' Declare variable.  
    On Error GoTo ErrorHandler ' Set up error handler.  
    PrintForm ' Print form.  
    Exit Sub  
ErrorHandler:  
    Msg = "The form can't be printed."  
    MsgBox Msg ' Display message.  
    Resume Next  
End Sub
```

## 11. PSet Method

Cú pháp : **[Object.]Pset [Step] (X, Y)[, Color]**

- Object : An object expression that evaluates to an object in the Applies To list. If object is omitted, the Form with the focus is assumed to be object.
- Step : A keyword specifying that the coordinates are relative to the current graphics position given by the CurrentX and CurrentY properties.
- (X, Y) : Single-precision values indicating the horizontal (x-axis) and vertical (y-axis) coordinates of the point to set.
- Color: Long integer value indicating the RGB color specified for point.

Tác dụng : tương tự như Point Method.

Ví dụ : vẽ các chấm điểm với màu sắc và vị trí ngẫu nhiên trên cửa sổ Form.

```
Private Sub Form_Click()  
    Dim CX, CY, Msg, XPos, YPos ' Declare variables.  
    ScaleMode = 3 ' Set ScaleMode to pixels.  
    DrawWidth = 5 ' Set DrawWidth.  
    ForeColor = QBColor(4) ' Set background to red.  
    FontSize = 24 ' Set point size.  
    CX = ScaleWidth / 2 ' Get horizontal center.  
    CY = ScaleHeight / 2 ' Get vertical center.  
    Cls ' Clear form.  
    Msg = "Chúc mừng năm mới!"  
    CurrentX = CX - TextWidth(Msg) / 2 ' Horizontal position.  
    CurrentY = CY - TextHeight(Msg) ' Vertical position.  
    Print Msg ' Print message.  
    Do  
        XPos = Rnd * ScaleWidth ' Get horizontal position.  
        YPos = Rnd * ScaleHeight ' Get vertical position.  
        PSet (XPos, YPos), QBColor(Rnd * 15) ' Draw confetti.  
        DoEvents ' Yield to other  
    Loop ' processing.  
End Sub
```

## 12. Refresh Method

Cú pháp : **[Object.]Refresh**

Tác dụng : cho phép "làm tươi" lại đối tượng, nghĩa là nó cho phép vẽ lại hình ảnh của Object.

Ví dụ :

```
Private Sub Form_Click ()
    Dim FNMA, I, Msg ' Declare variables.
    File1.Pattern = "TestFile.*" ' Set file pattern.
    For I = 1 To 8 ' Do eight times.
        FNMA = "TESTFILE." & I
        Open FNMA For Output As FreeFile ' Create empty file.
        File1.Refresh ' Refresh file list box.
        Close ' Close file.
    Next I
    Msg = "Choose OK to remove the created test files."
    MsgBox Msg ' Display message.
    Kill "TESTFILE.*" ' Remove test files.
    File1.Refresh ' Update file list box.
End Sub
```

### 13. Scale Method

Cú pháp : [Object.]Scale [(X1, Y1) - (X2, Y2)]

- Object : tên của đối tượng cần định lại hệ thống tọa độ.
- (X1, Y1) : tọa độ góc trên bên trái.
- (X2, Y2) : tọa độ góc phải bên dưới.

Tác dụng : qui định lại hệ thống tọa độ theo yêu cầu người sử dụng. Nếu không có (X1, Y1) và (X2, Y2) thì trả hệ thống tọa độ về giá trị ngầm định.

Ví dụ :

```
Private Sub Tile_Click()
    Dim I, OldFontSize ' Declare variables.
    Width = 8640: Height = 5760 ' Set form size in twips.
    Move 100, 100 ' Move form origin.
    AutoRedraw = -1 ' Turn on AutoRedraw.
    OldFontSize = FontSize ' Save old font size.
    BackColor = QBColor(7) ' Set background to gray.
    Scale (0, 110)-(130, 0) ' Set custom coordinate system.
    For I = 100 To 10 Step -10
        Line (0, I)-(2, I) ' Draw scale marks every 10 units.
        CurrentY = CurrentY + 1.5 ' Move cursor position.
        Print I ' Print scale mark value on left.
        Line (ScaleWidth - 2, I)-(ScaleWidth, I)
        CurrentY = CurrentY + 1.5 ' Move cursor position.
        CurrentX = ScaleWidth - 9
        Print I ' Print scale mark value on right.
    Next I
    ' Draw bar chart.
    Line (10, 0)-(20, 45), RGB(0, 0, 255), BF ' First blue bar.
    Line (20, 0)-(30, 55), RGB(255, 0, 0), BF ' First red bar.
    Line (40, 0)-(50, 40), RGB(0, 0, 255), BF
    Line (50, 0)-(60, 25), RGB(255, 0, 0), BF
    Line (70, 0)-(80, 35), RGB(0, 0, 255), BF
```

```

Line (80, 0)-(90, 60), RGB(255, 0, 0), BF
Line (100, 0)-(110, 75), RGB(0, 0, 255), BF
Line (110, 0)-(120, 90), RGB(255, 0, 0), BF
CurrentX = 18: CurrentY = 100 ' Move cursor position.
FontSize = 14 ' Enlarge font for title.
Print "Widget Quarterly Sales" ' Print title.
FontSize = OldFontSize ' Restore font size.
CurrentX = 27: CurrentY = 93 ' Move cursor position.
Print "Planned Vs. Actual" ' Print subtitle.
Line (29, 86)-(34, 88), RGB(0, 0, 255), BF ' Print legend.
Line (43, 86)-(49, 88), RGB(255, 0, 0), BF
Scale
End Sub

```

Chú ý : ta có thể thay đổi đơn vị đo trong Visual Basic bằng cách thay đổi trị số của ScaleMode trong bộ thuộc tính Properties.

#### 14. SetFocus Method

Cú pháp : **[Object.]SetFocus**

Tác dụng : cho phép tham chiếu đến Object có tên được chỉ định để thực hiện các thay đổi trên đó nếu có.

Ví dụ : Vehinh.SetFocus

#### 15. TextHeight và TextWidth Methods

Cú pháp : **[Object.]TextHeight (String)**

**[Object.]TextWidth (String)**

- Object : tên của đối tượng đã được ấn định kích cỡ Font chữ mà ta dựa vào đó để tính chiều cao và chiều rộng của đoạn văn bản cần thể hiện.
- String : nội dung chuỗi ký tự mà Method sẽ tính toán chiều cao và chiều rộng.

Tác dụng : tính toán và trả về kết quả là chiều cao và chiều rộng của String.

Ví dụ :

```

Private Sub Inchu_Click()
Dim HalfWidth, HalfHeight, Msg ' Declare variable.
AutoRedraw = -1 ' Turn on AutoRedraw.
BackColor = QBColor(4) ' Set background color.
ForeColor = QBColor(15) ' Set foreground color.
Msg = "Visual Basic" ' Create message.
FontSize = 48 ' Set font size.
HalfWidth = TextWidth(Msg) / 2 ' Calculate one-half width.
HalfHeight = TextHeight(Msg) / 2 ' Calculate one-half height.
CurrentX = ScaleWidth / 2 - HalfWidth ' Set X.
CurrentY = ScaleHeight / 2 - HalfHeight ' Set Y.
Print Msg ' Print message.
End Sub

```

### III. Hàm

#### 1. Giới thiệu

Trong Visual Basic đã xây dựng sẵn các hàm để phục vụ cho việc xử lý dữ liệu một cách dễ dàng và nhanh chóng. Trong phần này ta xét một số hàm thường dùng.

#### 2. Các hàm xử lý chuỗi

##### a. Tìm chiều dài chuỗi: **LEN(String)**

Trả về kết quả là số ký tự có trong String.

Ví dụ : LEN("ABCD") trả về kết quả là 4.

##### b. Chuyển sang chữ thường : **LCase(String)** hoặc **Lcase\$(String)**

Trả về kết quả là chuỗi ký tự mới sau khi đổi chuỗi cũ sang chữ thường. Nếu có dấu \$ thì trả về kết quả thuộc kiểu dữ liệu Varian nếu có \$ kết quả trả về kiểu String.

Ví dụ : LCase("ABCD") trả về kết quả là abcd.

##### c. Chuyển sang chữ in : **UCase(String)** hoặc **Ucase\$(String)**

Trả về kết quả là chuỗi ký tự mới sau khi đổi chuỗi cũ sang chữ in. Nếu có dấu \$ thì trả về kết quả thuộc kiểu dữ liệu Varian nếu có \$ kết quả trả về kiểu String.

Ví dụ : UCase("abcd") trả về kết quả là ABCD.

##### d. Lấy các ký tự bên trái : **Left(String,n)** hoặc **Left\$(String,n)**

Trả về kết quả là chuỗi ký tự mới gồm n ký tự bên trái của chuỗi cũ.

Ví dụ : Left("ABCD",2) trả về kết quả là AB

##### e. Lấy các ký tự bên phải: **Right(String, n)** hoặc **Right\$(String,n)**

Trả về kết quả là chuỗi ký tự mới gồm n ký tự bên phải của chuỗi cũ.

Ví dụ : Right("ABCD",2) trả về kết quả là CD

##### f. Lấy nhóm ký tự bất kỳ: **Mid(String,m,n)** hoặc **Mid\$(String,m,n)**

Trả về kết quả là chuỗi ký tự mới gồm n ký tự bắt đầu từ ký tự thứ m của chuỗi cũ.

Ví dụ : Mid("ABCD",2,2) trả về kết quả là BC

##### g. Bỏ các ký tự trống: **Trim(String)** hoặc **Trim\$(String)**

Trả về kết quả là chuỗi ký tự mới sau khi đã vớt bỏ các ký tự trống ở hai đầu chuỗi cũ.

Ví dụ : Trim(" AB ") trả về kết quả là "AB"

##### h. Bỏ các ký tự trống bên trái: **LTrim(String)** hoặc **LTrim\$(String)**

Trả về kết quả là chuỗi ký tự mới sau khi đã vớt bỏ các ký tự trống bên trái của chuỗi cũ.

Ví dụ : LTrim(" AB ") trả về kết quả là "AB "

##### i. Bỏ các ký tự trống bên phải: **RTrim(String)** hoặc **RTrim\$(String)**

Trả về kết quả là chuỗi ký tự mới sau khi đã vớt bỏ các ký tự trống bên phải của chuỗi cũ.

Ví dụ : RTrim(" AB ") trả về kết quả là " AB"

**j. Đổi mã số sang ký tự: *Chr(mã số)* hoặc *Chr\$(mã số)***

Trả về kết quả là một ký tự tương ứng với mã số trong bảng mã ANSI. Mã số là một số nguyên từ 0 đến 255.

Ví dụ : Chr(65) trả về kết quả là "A"

**k. Đổi ký tự sang mã số: *Asc(Ký tự)***

Trả về kết quả là một số kiểu Integer tương ứng với ký tự trong bảng mã ANSI.

Ví dụ : Asc("A") trả về kết quả là 65.

**l. Đổi chuỗi sang số: *Val(biểu thức chuỗi)***

Trả về kết quả là một số sau khi đổi chuỗi dạng số (kiểu String) sang giá trị số.

Ví dụ : Val("123") + Val("213") trả về kết quả là 336

**m. Đổi số sang chuỗi: *Str\$(biểu thức số)***

Trả về kết quả là một chuỗi ký tự sau khi đổi số sang.

Ví dụ : Str(123) + Str(213) trả về kết quả là "123213"

**n. Định dạng chuỗi: *Format\$(biểu thức [, dạng])***

Trả về kết quả là một chuỗi ký tự được định dạng theo một khuôn mẫu cho trước. Biểu thức ở đây có thể là số hoặc chuỗi.

- Các ký tự định dạng số :

- # : hiển thị số nếu có còn không thì không hiện gì cả.
- 0 : hiển thị số nếu có còn không thì xuất hiện ký tự 0.
- . : hiển thị dấu chấm ở vị trí khai báo.
- , : hiển thị dấu phẩy ở vị trí khai báo.
- % : nhân biểu thức với 100 rồi xuất hiện dấu %.

Ví dụ :

So! = 1234.5	
Format(so, "#.###")	kết quả 1234.5
Format(so, "###,###")	kết quả 1,234.5
Format(so, "0.000")	kết quả 1234.5000
Format(so, "0%")	kết quả 1234500%
Format(so, "\$0.00")	kết quả \$1234.50

- Các ký tự định dạng chuỗi ký tự :

- & : hiển thị ký tự nếu có còn không thì không hiện gì cả.
- & : hiển thị ký tự nếu có còn không thì hiện lên một ký tự trắng.
- < : đổi chuỗi sang chữ thường.
- > : đổi chuỗi sang chữ in.

Ví dụ :

Format("visual basic", ">")	trả về "VISUAL BASIC"
Format("VISUAL BASIC", ">")	trả về "visual basic"

**o. Tìm chuỗi con: *InStr*(\$)([số,] chuỗi 1, chuỗi 2[, so sánh])**

Trong đó :

- Số : nếu có thì nó qui định vị trí bắt đầu tìm kiếm. Không có thì tìm từ đầu.
- So sánh : là qui định phương thức tìm. Nếu so sánh là giá trị 1 thì không phân biệt chữ in với chữ thường, nếu giá trị 0 thì có phân biệt chữ in với chữ thường.
- Chuỗi 1 : chuỗi mẹ. Đây là chuỗi có thể chứa chuỗi cần tìm.
- Chuỗi 2 : chuỗi con. Đây là chuỗi cần tìm xem có được chứa trong chuỗi 1 hay không.

Hàm này trả về kết quả là giá trị số. Nếu bằng 0 nghĩa là không tìm thấy, nếu một số lớn thì không thì đó là vị trí xuất hiện chuỗi 2 trong chuỗi 1.

Ví dụ : `InStr("I Love You", "Love")` trả về kết quả là 3

`InStr("I Love You", "love", 0)` trả về kết quả 0.

**3. Các hàm xử lý số :**

<code>SIN(góc)</code>	Tính sin của góc.
<code>COS(góc)</code>	Tính Cosin của một góc
<code>TAN(góc)</code>	Tính Tang của một góc
<code>ATAN(số)</code>	Tính Arctang của một góc
<code>EXP(số)</code>	Expenential
<code>LOG(số)</code>	Logarithm
<code>CCUR(số)</code>	Chuyển đổi một số về kiểu Currency
<code>CINT(số)</code>	Chuyển đổi một số về kiểu Integer
<code>CLNG(số)</code>	Chuyển đổi một số về kiểu Long
<code>CSNG(số)</code>	Chuyển đổi một số về kiểu Single
<code>CDBL(số)</code>	Chuyển đổi một số về kiểu Double
<code>FIX(số)</code>	Bỏ phần thập phân để đổi thành số nguyên
<code>INT(số)</code>	Qui tròn về số nguyên.
<code>RND[(số)]</code>	Tạo một số ngẫu nhiên.
<code>ABS(số)</code>	Trị tuyệt đối
<code>SGN(số)</code>	Dấu âm/dương của một con số
<code>SQR(số)</code>	Lấy căn bậc hai.



## CHƯƠNG 5. THIẾT KẾ GIAO DIỆN

### I. Giới thiệu

Visual Basic cung cấp một giao diện đồ họa người dùng hoàn chỉnh và giúp cho người lập trình rất nhiều trong việc thiết kế giao diện khi phát triển các phần mềm. Người lập trình chỉ cần sử dụng và khai báo hợp lý các đối tượng để thiết kế giao diện và trên cơ sở đó máy tính sẽ tự phát sinh ra mã nguồn tương ứng.

Trong chương này, chúng tôi chỉ chọn giới thiệu những loại đối tượng cần thiết và tương đối phức tạp khi thiết kế là Listbox và Combobox. Ngoài ra, chúng tôi cũng giới thiệu cách thức để người sử dụng tự tạo ra các đối tượng phù hợp với yêu cầu thực tế khi thiết kế giao diện.

### II. Dùng list control

Có hai loại List controls dùng trong VB6 là Listbox và Combobox. Cả hai đều hiển thị một danh sách các giá trị để người sử dụng có thể lựa chọn mà không cần phải nhập từ bàn phím. Điểm khác biệt khi thiết kế giao diện là Listbox có thể nằm trên nhiều dòng còn Combobox chỉ hiển thị trên một dòng.

Listbox có thể chiếm một khung chữ nhật, nếu chiều dài không đủ để hiển thị tất cả mọi dòng thì Listbox tự động cho ta một thanh cuộn đứng để cho biết còn có nhiều dòng bị che và ta có thể xem các dòng ấy bằng cách dùng phím hoặc chuột để thay đổi các dòng.

Combobox chỉ hiển thị trên một dòng nhưng ta có thể chọn để xem những dòng khác bằng cách click chuột vào dấu hiệu ▼.



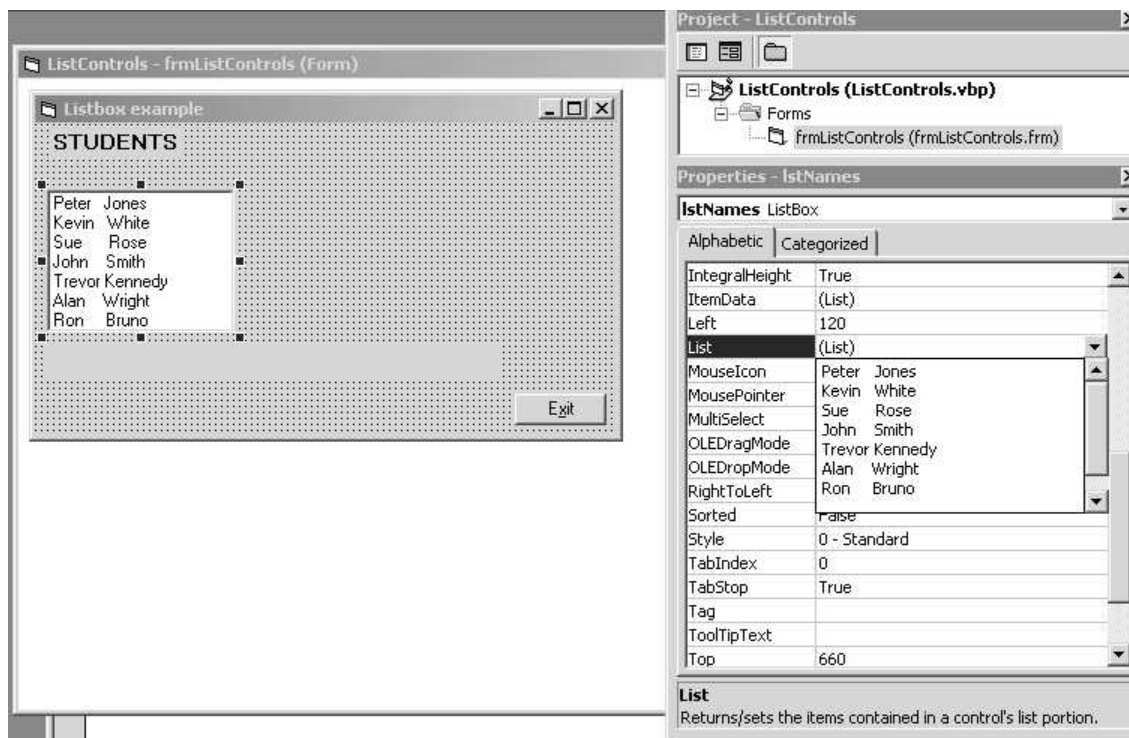
Listbox có rất nhiều công dụng vì nó rất uyển chuyển khi sử dụng. Trong bài này chúng ta sẽ xem xét một số ứng dụng của Listbox khi thiết kế giao diện:

- Hiển thị nhiều sự lựa chọn để người sử dụng có thể chọn bằng cách click hay drag-drop.
- Chọn lựa một giá trị trong danh sách.
- Dùng để hiển thị các sự kiện.
- Dùng để tìm kiếm hoặc xử lý văn bản.
- Dùng làm hàng đợi.

## 1. Listbox

### a. *Hiển thị nhiều sự lựa chọn*

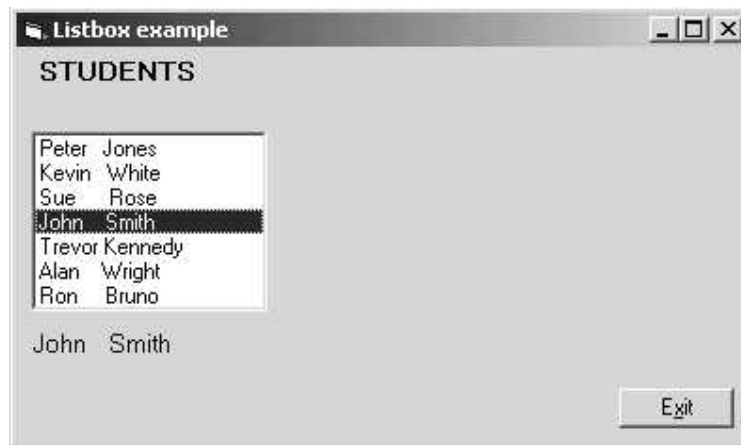
Ta thử bắt đầu viết một chương trình gồm có một Listbox tên **lstNames** nằm trong một Form. Trong **lstNames** ta đánh vào tên của bảy người, mỗi lần xuống dòng nhớ đánh **Ctrl-Enter**, thay vì chỉ **Enter**, nếu không VB6 ngầm hiểu đã đánh xong nên tự đóng cửa sổ thuộc tính List. Các tên này là những dòng sẽ hiện ra trong Listbox khi ta bắt đầu chạy chương trình.



Ngoài **lstNames** ta cho thêm một Label với Caption **STUDENTS** để trang trí, và một Label khác tên **lblName**. Mỗi khi người sử dụng click lên dòng tên nào thì sẽ hiển thị dòng tên ấy trong **lblName**. Sau cùng ta cho vào một CommandButton tên **CmdExit** để cho dừng chương trình. Ta sẽ có chương trình như sau:

```
Private Sub lstNames_Click()  
    lblName.Caption = lstNames.List(lstNames.ListIndex)  
End Sub  
Private Sub CmdExit_Click()  
    End  
End Sub
```

Giả sử ta click vào tên John Smith trên Listbox, ta sẽ thấy tên ấy cũng được hiển thị trong Label **lblName**.



Trong ví dụ này, Listbox lstNames có 7 dòng (**Items**). Số Items này là thuộc tính **ListCount** của Listbox. Các Items của Listbox được đếm từ **0 đến ListCount-1**. Trong trường hợp này là từ 0 đến 6.

Khi người sử dụng click lên một dòng, Listbox sẽ phát sinh **Event lstNames\_Click**. Lúc bấy giờ ta có thể biết được người sử dụng vừa mới Click dòng nào bằng cách hỏi thuộc tính **ListIndex** của lstNames, nó sẽ có giá trị từ 0 đến ListCount-1. Lúc chương trình mới chạy, chưa ai Click lên Item nào của Listbox thì ListIndex = -1.

Những Items trong Listbox được xem như một mảng chuỗi ký tự. Mảng này được gọi là **List**. Do đó, ta nói đến Item thứ nhất của Listbox lstNames bằng cách viết **lstNames.List(0)**, và tương tự như vậy, Item cuối cùng là **lstNames.List(lstNames.ListCount-1)**.

Ta có thể nói đến item vừa được Clicked bằng hai cách:

- lstNames.List(lstNames.ListIndex)
- lstNames.text.

#### **b. Ghi nội dung của Listbox**

Bây giờ để lưu trữ nội dung của lstNames, ta thêm một CommandButton tên CmdSave. Ta sẽ viết code để khi người sử dụng click nút CmdSave chương trình sẽ mở một tập tin (dạng text) và viết mọi phần tử của lstNames vào đó:

```
Private Sub CmdSave_Click()
    Dim i, FileName, FileNumber
    FileName = App.Path
    ' Make sure FileName ends with a backslash
    If Right(FileName, 1) <> "\" Then FileName = FileName & "\"
    FileName = FileName & "MyList.txt" 'output text file MyList.txt
    ' Obtain an available filename from the operating system
    FileNumber = FreeFile
    ' Open the FileName as an output file
    Open FileName For Output As FileNumber
    ' Now iterate through each item of lstNames
    For i = 0 To lstNames.ListCount - 1
        ' Write the List item to file
        Print #FileNumber, lstNames.List(i)
    Next
    Close FileNumber ' Close the output file
End Sub
```

**App** là một Object đặc biệt đại diện cho chính chương trình đang chạy. Ở đây ta dùng thuộc tính **Path** để biết lúc chương trình đang chạy thì mô-đun EXE của nó nằm ở đâu. Lý do là ta thường để các tập tin liên hệ cần thiết cho chương trình nằm ở ngay trong thư mục của chương trình hay trong một thư mục con, chẳng hạn như **data**, **logs**,... App còn có một số thuộc tính khác cũng rất có ích như **PrevInstance**, **Title**, **Revision**... Nếu mới khởi động một chương trình mà thấy App.PrevInstance = True thì lúc bấy giờ cũng có một bản sao khác của chương trình đang chạy. Nếu cần, ta kết thúc chương trình này để tránh chạy 2 bản sao của chương trình cùng một lúc.

App.Title và App.Revision cho ta tin tức về Title và Revision của chương trình đang chạy. Để viết ra một tập tin định dạng văn bản (text file) ta cần phải mở nó trong chế độ output và khai báo từ đây trở đi sẽ dùng một chữ số (FileNumber) để đại diện tập tin thay vì dùng chính tên tập tin. Để tránh dùng một FileNumber đã hiện hữu, tốt nhất ta đề nghị hệ điều hành cung cấp một giá trị số chưa ai dùng bằng cách gọi **Function FreeFile**. Chữ số FileNumber này còn được gọi là **FileHandle**. Sau khi ta Close FileNumber chữ số này được giải phóng và hệ điều hành sẽ có thể dùng nó lại cho lần gọi sau.

Do đó, chúng ta phải tránh gọi FreeFile liên tiếp hai lần vì hệ điều hành sẽ cho chúng ta cùng một chữ số. Tức là, sau khi gọi FreeFile phải dùng nó ngay bằng cách mở một File rồi mới gọi FreeFile lần kế để có một chữ số khác.

Để ý cách dùng chữ **Input**, **Output** cho tập tin là tương đối với vị trí của chương trình (nó nằm trong bộ nhớ của máy tính). Nếu từ trong bộ nhớ viết ra đĩa cứng thì gọi là Output. Ngược lại, nếu đọc từ một tập tin nằm trên đĩa cứng vào bộ nhớ cho chương trình ta thì gọi là Input.

### c. *Tải một text file vào Listbox*

Trong phần này, thay vì đánh các Items của Listbox vào thuộc tính List của lstNames ta có thể tạo các giá trị của lstNames bằng cách đọc các Items từ một tập tin văn bản. Ta thử thêm một CommandButton tên CmdLoad. Ta sẽ viết code để khi người sử dụng nhấn nút CmdLoad chương trình sẽ mở một tập tin đầu vào dạng văn bản và đọc từng dòng để bỏ vào lstNames:

```
Private Sub CmdLoad_Click()
    Dim i, FileName, FileNumber, anItem
    ' Obtain Folder where this program's EXE file resides
    FileName = App.Path
    ' Make sure FileName ends with a backslash
    If Right(FileName, 1) <> "\" Then FileName = FileName & "\"
    FileName = FileName & "MyList.txt"
    ' Obtain an available filenumber from the operating system
    FileNumber = FreeFile
    ' Open the FileName as an input file
    Open FileName For Input As FileNumber
    lstNames.Clear ' Clear the Listbox first
    ' Now read each line until reaching End-Of-File
    Do While NOT EOF(FileNumber)
        Line Input #FileNumber, anItem ' Read a line from the file
        lstNames.AddItem anItem ' Add this item to the lstNames
    Loop
    Close FileNumber ' Close the input file
End Sub
```

Để đọc từ một tập tin ta cần phải mở nó trong chế độ output. Trước khi làm đầy lstNames ta cần phải xóa tất cả mọi items có sẵn bên trong. Để thực hiện việc đó ta dùng phương thức **Clear** của Listbox.

Sau đó ta dùng phương thức **AddItem** để cho thêm từng dòng vào trong Listbox. Ở chế độ mặc định, nếu ta không chỉ rõ chèn vào ở dòng nào thì AddItem sẽ chèn Item mới vào phần tử cuối của Listbox.

Nếu muốn chèn dòng mới vào ngay trước item thứ 5 (ListIndex = 4), ta viết:

```
lstNames.AddItem newItemString, 4 ' newItemString contains  
' To insert a new Item at the beginning of the Listbox, write:  
lstNames.AddItem newItemString, 0
```

Lưu ý là mỗi lần chúng ta Add một Item vào Listbox thì ListCount của Listbox tăng 1. Muốn xóa một item từ Listbox ta dùng phương thức **RemoveItem**, ví dụ như muốn xóa item thứ ba (ListIndex=2) của lstNames, ta viết:

```
lstNames.RemoveItem 2
```

Mỗi lần chúng ta xóa một phần tử từ Listbox thì ListCount của Listbox giảm đi một đơn vị. Do đó, nếu chúng ta dùng Test dựa vào ListCount của một ListBox để nhảy ra khỏi một Loop thì phải lưu ý tránh làm cho giá trị ListCount thay đổi trong Loop vì thêm hay bớt phần tử.

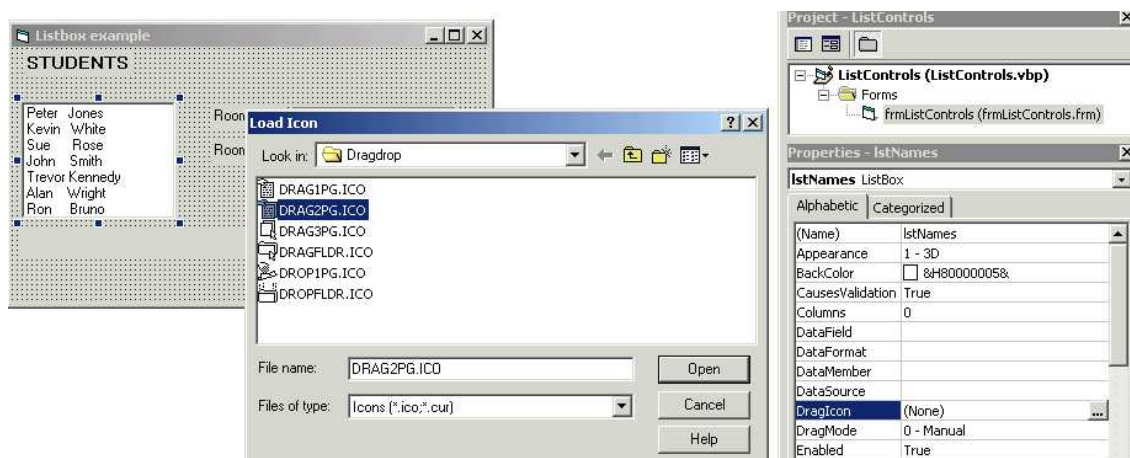
Ta đọc từng dòng của một Text file bằng cách dùng **Line Input #FileNumber**. Khi đọc đến cuối tập tin, hệ thống sẽ cho ta giá trị EOF(FileNumber) = True. Ta dùng giá trị ấy để cho chương trình nhảy ra khỏi While.. Loop.

Câu **Do While NOT EOF(FileNumber)** có nghĩa **khi chưa đến cuối của Text File đại diện bởi FileNumber** thì đọc từng dòng và bỏ vào Listbox.

## 2. Drag-Drop

Ta đã xem qua Click Event của Listbox. Bây giờ để dùng Drag-Drop cho Listbox chúng ta hãy đặt 2 Labels mới lên Form. Cái thứ nhất tên gì cũng được nhưng có Caption là Room A. Hãy gọi nhãn thứ hai là lblRoom và cho thuộc tính BorderStyle của nó bằng Fixed Single. Kế đến chọn cả hai nhãn rồi chọn copy và paste lên Form. VB6 sẽ cho chúng ta hai nhãn lblRoom.

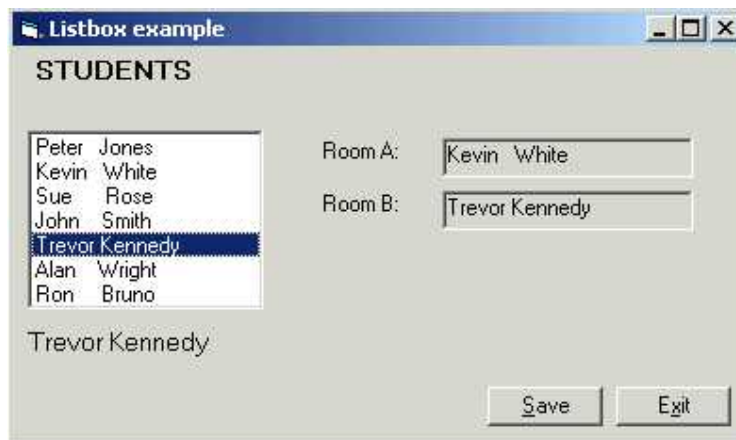
Để cho lstNames một DragIcon, chúng ta click lstNames, chọn thuộc tính DragIcon để pop-up một hộp thoại cho chúng ta chọn một dragdrop icon từ thư mục C:\Program Files\Microsoft Visual Studio\Common\Graphics\Icons\Dragdrop, chẳng hạn như DRAG2PG.ICO:



Ta sẽ dùng Event MouseDown của lstNames để pop-up DragIcon hình 2 trang giấy cho UserDrag nó qua bên phải rồi bỏ xuống lên một trong hai lblRoom. Khi DragIcon rơi lên lblRoom, lblRoom sẽ generate Event DragDrop. Ta sẽ dùng Event DragDrop này để gán thuộc tính Text của Source (tức là lstNames, mục control từ nó phát xuất Drag action) vào thuộc tính Caption của lblRoom. Lưu ý vì ở đây ta dùng cùng một tên cho cả hai lblRoom nên chỉ cần viết mã chương trình ở một chỗ để quản lý Event DragDrop.

```
Private Sub lstNames_MouseDown(Button As Integer, Shift As Integer, X As Single, Y As Single)
    ' Start Pop-up DragIcon and start Drag action
    lstNames.Drag
End Sub
Private Sub lblRoom_DragDrop(Index As Integer, Source As Control, X As Single, Y As Single)
    ' Assign Property Text of Source (i.e. lstNames) to Caption
    lblRoom(Index).Caption = Source.Text
End Sub
```

Kết quả sau khi Drag hai tên từ Listbox qua Labels là như sau:



### 3. Dùng thuộc tính Sorted

Trong ví dụ trên ta có thể quyết định vị trí của một Item mới khi ta chèn nó vào Listbox. Đôi khi ta muốn các Items của Listbox được tự động sắp theo thứ tự Alphabet. Chúng ta có thể thiết lập thuộc tính **Sorted = True** để thực hiện chuyện này. Có một giới hạn là chúng ta phải cho thuộc tính Sorted một giá trị (True hay False) trong lúc thiết kế. Thuộc tính này không thể thay đổi khi chạy chương trình.

Giả sử ta muốn sắp xếp các Items của một Listbox khi cần. Vậy thì ta làm sao? Giải pháp rất đơn giản. Chúng ta tạo một Listbox tên lstTemp chẳng hạn. Cho nó thuộc tính Visible=False (để không hiển thị) và thuộc tính Sorted=True. Khi cần sắp xếp lstNames chẳng hạn, ta chép nội dung của lstNames bỏ vào lstTemp, tiếp đến xóa lstNames rồi chép nội dung (đã được sắp) của lstTemp trở lại lstNames.

Lưu ý là ta có thể AddItem vào một Listbox với thuộc tính Sorted=True, nhưng không thể xác định chèn Item vào trước dòng nào, vì vị trí của các Items do Listbox quyết định khi nó sắp xếp các Items.

Ta hãy cho thêm vào Form một CommandButton mới tên **CmdSort** và viết code cho Event Click của nó như sau:

```
Private Sub CmdSort_Click()
```

```

Dim i
lstTemp.Clear ' Clear temporary Listbox
' Iterate though every item of lstNames
For i = 0 To lstNames.ListCount - 1
    ' Add the lstNames item to lstTemp
    lstTemp.AddItem lstNames.List(i)
Next
lstNames.Clear ' Clear lstNames
' Iterate though every item of lstTemp
For i = 0 To lstTemp.ListCount - 1
    ' Add the lstTemp item to lstNames
    lstNames.AddItem lstTemp.List(i)
Next
lstTemp.Clear ' Tidy up - clear temporary Listbox
End Sub

```

Trong trường hợp ta muốn có tùy chọn để sắp xếp các tên theo FirstName hay Surname thì vẫn có thể dùng sorted Listbox vô hình tên lstTemp.

Chúng ta hãy đặt lên phía trên lstName hai Labels mới tên lblFirstName và lblSurName và cho chúng Caption "FirstName" và "SurName".

Từ đây ta tải tập tin "MyList.txt" vào lstNames bằng cách nhấp chuột vào nút CmdLoad mà không sửa thuộc tính List của lstNames để nhập các phần tử lúc thiết kế nữa. Ngoài ra ta dùng dấu phẩy (,) để tách FirstName khỏi SurName trong mỗi tên chứa trong file MyList.txt. Nội dung của tập tin MyList.txt bây giờ trở thành như sau:

```

Peter, Jones
Kevin, White
Sue, Rose
John, Smith
Trevor, Kennedy
Alan, Wright
Ron, Bruno

```

Ta sẽ sửa code trong Sub CmdLoad\_Click lại để khi chèn tên vào lstNames, FirstName và SurName mỗi thứ chiếm 10 ký tự.

Để các chữ trong Items của lstNames sắp dòng ngay ngắn ta đổi Font của lstNames ra Courier New. Courier New là một loại phong chữ mà chiều ngang của tất cả các chữ là như nhau trong khi hầu hết các phong chữ khác như Arial, Times Roman ... có độ rộng các ký tự là khác nhau (Proportional Spacing).

Mã mới của Sub CmdLoad\_Click trở thành như sau:

```

Private Sub CmdLoad_Click()
    Dim i, Pos
    Dim FileName, FileNumber, anItem
    Dim sFirstName As String*10 ' fixed length string of 10 chars
    Dim sSurName As String * 10 ' fixed length string of 10 chars
    ' Obtain Folder where this program's EXE file resides
    FileName = App.Path
    ' Make sure FileName ends with a backslash
    If Right(FileName, 1) <> "\" Then FileName = FileName & "\"
    FileName = FileName & "MyList.txt"
    ' Obtain an available filename from the operating system

```

```

    FileNumber = FreeFile
    ' Open the FileName as an input file , using FileNumber
    Open FileName For Input As FileNumber
    lstNames.Clear ' Clear the Listbox first
    ' Now read each line until reaching End-Of-File
    Do While Not EOF(FileNumber)
        Line Input #FileNumber, anItem ' Read a line from the file
        ' Now separate FirstName from SurName
        Pos = InStr(anItem, ",") ' Locate the comma ","
        ' The part before "," is FirstName
        sFirstName = Left(anItem, Pos - 1)
        sFirstName = Trim(sFirstName) ' Trim off any blank spaces
        ' The part after "," is SurName
        sSurName = Mid(anItem, Pos + 1)
        sSurName = Trim(sSurName) ' Trim off any blank spaces
        lstNames.AddItem sFirstName & sSurName
        ' Add this item to the bottom of lstNames
    Loop
    Close FileNumber ' Close the input file
End Sub

```

Vì FirstName nằm ở bên trái của mỗi Item nên sắp xếp theo FirstName cũng giống như sắp xếp cả Item. Việc ấy ta đã làm bằng Sub CmdSort\_Click rồi, do đó khi người sử dụng click Label lblFirstName ta chỉ cần gọi CmdSort\_Click như sau:

```

Private Sub lblFirstName_Click()
    CmdSort_Click
End Sub

```

Để sắp xếp theo SurName ta cần phải tạm thời để SurName qua bên trái của Item trước khi bỏ vào lstTemp. Ta thực hiện chuyện này bằng cách hoán chuyển vị trí của FirstName và SurName trong Item trước khi bỏ vào lstTemp. Sau đó, khi copy các Items từ lstTemp để đặt vào lại lstNames ta lại nhớ hoán chuyển FirstName và SurName để chúng nằm đúng lại vị trí. Đoạn mã để sắp xếp tên theo SurName cũng giống như CmdSort\_Add nhưng sửa đổi chút ít như sau:

```

Private Sub lblSurName_Click()
    Dim i, anItem
    Dim sFirstName As String*10 'fixed length string of 10 chars
    Dim sSurName As String * 10 ' fixed length string of 10 chars
    lstTemp.Clear ' Clear temporary Listbox
    ' Iterate though every item of lstNames
    For i = 0 To lstNames.ListCount - 1
        anItem = lstNames.List(i)
        ' Identify FistName and SurName
        sFirstName = Left(anItem, 10)
        sSurName = Mid(anItem, 11)
        ' Swap FirstName/SurName positions before adding to lstTemp
        lstTemp.AddItem sSurName & sFirstName
    Next
    lstNames.Clear ' Clear lstNames
    ' Iterate though every item of lstTemp
    For i = 0 To lstTemp.ListCount - 1
        anItem = lstTemp.List(i)

```

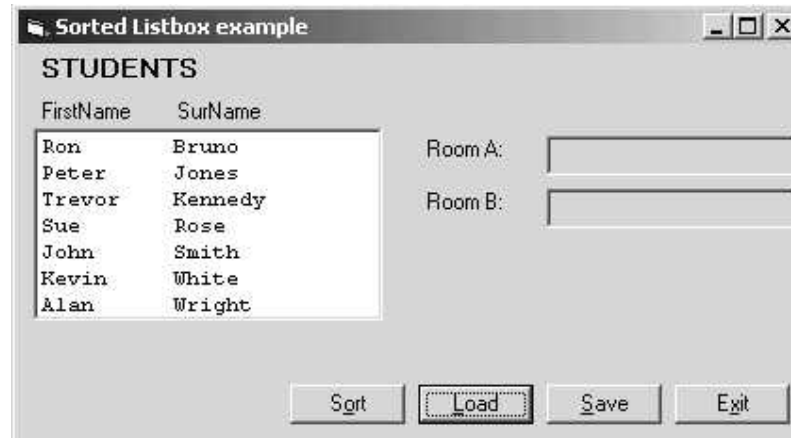


```

        sSurName = Left(anItem, 10) ' SurName now is on the left
        sFirstName = Mid(anItem, 11)
        ' Add FirstName/SurName in correct positions to lstNames
        lstNames.AddItem sFirstName & sSurName
    Next
    lstTemp.Clear ' Tidy up - clear temporary Listbox
End Sub

```

Các Items trong lstNames đã được sắp xếp theo SurName hiện ra như sau:



Ta sửa Sub CmdSave\_Click để Save Items theo trật tự sắp xếp mới nếu cần:

```

Private Sub CmdSave_Click()
    Dim i, FileName, FileNumber, anItem
    ' Obtain Folder where this program's EXE file resides
    FileName = App.Path
    ' Make sure FileName ends with a backslash
    If Right(FileName, 1) <> "\" Then FileName = FileName & "\"
    ' Call Output filename "MyList.txt"
    FileName = FileName & "MyList.txt"
    ' Obtain an available filename from the operating system
    FileNumber = FreeFile
    ' Open the FileName as an output file, using FileNumber
    Open FileName For Output As FileNumber
    ' Now iterate through each item of lstNames
    For i = 0 To lstNames.ListCount - 1
        anItem = lstNames.List(i)
        anItem=Trim(Left(anItem, 10)) & "," & Trim(Mid(anItem, 11))
        ' Write the List item to file. Make sure you use symbol #
in front of FileNumber
        Print #FileNumber, anItem
    Next
    Close FileNumber ' Close the output file
End Sub

```

### III. Tự tạo các đối tượng (Object)

Từ trước đến giờ, ta lập trình VB6 bằng cách thiết kế các Forms rồi viết codes để xử lý các Events của những controls trên Form khi người sử dụng click một Button hay Listbox, .v.v.. Nói chung, cách ấy cũng hữu hiệu để triển khai chương trình, nhưng nếu ta có thể hưởng được các lợi ích sau đây thì càng tốt hơn :

- Dùng lại được code đã viết trước đây trong một dự án khác
- Dễ nhận diện được một lỗi (error) phát xuất từ đâu
- Dễ triển khai một dự án lớn bằng cách phân phối ra thành nhiều dự án nhỏ
- Dễ bảo trì

Lập trình theo hướng đối tượng là thiết kế các bộ phận phần mềm của chương trình, gọi là **Objects** sao cho mỗi bộ phận có thể tự lo liệu công tác của nó giống như một module **làm việc độc lập**. Câu hỏi đặt ra là các **Sub** hay **Function** mà chúng ta đã từng viết để xử lý từng giai đoạn trong chương trình có thể đảm trách vai trò của một module độc lập không?

Có một cách định nghĩa khác cho Object là một Object gồm có data structure và các Subs/Functions làm việc trên các data ấy. Thông thường, khi ta dùng Objects không cần giám sát chúng thực hiện như thế nào, ngược lại nếu khi có sự cố gì thì ta muốn chúng báo cáo cho ta biết.

Trong VB6, các Forms, Controls hay ActiveX là những Objects mà ta vẫn sử dụng. Lấy ví dụ như Listbox. Một Listbox tự quản lý các items hiển thị bên trong nó. Ta biết listbox List1 đang có bao nhiêu items bằng cách hỏi List1.ListCount. Ta biết item nào vừa mới được selected bằng cách hỏi List1.ListIndex. Ta thêm một item vào listbox bằng cách gọi phương thức AddItem của List1, ..v.v.. Nói cho đúng ra, Object là một thực thể của một Class. Nếu Listbox là một Class thì List1, List2 là các thực thể của Listbox.

Ngay cả một form tên frmMyForm mà ta viết trong VB6 chẳng hạn, nó cũng là một Class. Thông thường ta dùng thẳng frmMyForm như sau:

```
frmMyForm.Show
```

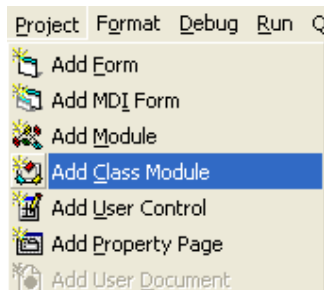
Trong trường hợp này thật ra frmMyForm tuy là một Class nhưng được dùng y như một Object. Nếu cần thiết, ta có thể tạo ra hai, ba Objects của Class frmMyForm cùng một lúc như trong ví dụ sau:

```
Dim firstForm As frmMyForm
Dim secondForm As frmMyForm
Set firstForm = New frmMyForm
Set secondForm = New frmMyForm
firstForm.Show
secondForm.Show
```

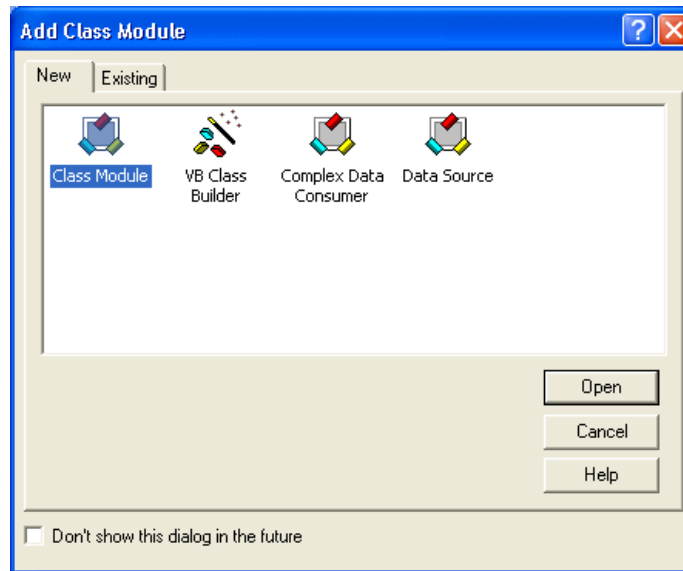
Trong ví dụ trên ta khai báo firstForm và secondForm là những Objects của Class frmMyForm. Sau đó ta làm nên (**instantiate**) các Objects firstForm và secondForm bằng statements **Set... = New...** firstForm và secondForm còn được gọi là các **instances** của Class frmMyForm. Class giống như cái khuôn, còn Objects giống như những cái bánh làm từ khuôn ấy. Chắc chúng ta đã để ý thấy trong VB6 từ dùng hai từ Class và Object lẫn lộn nhau. Điều này cũng không quan trọng, miễn là chúng ta nắm vững ý nghĩa của chúng.

VB6 có hỗ trợ **Class** mà ta có thể triển khai và instantiate các Objects của nó khi dùng. Một Class trong VB6 có chứa **data** riêng của nó, có những **Subs** và **Functions** mà ta có thể gọi. Ngoài ra, Class còn có thể Raise **Events**, tức là báo cho ta biết khi chuyện gì xảy ra bên trong nó. Cũng giống như Event Click của CommandButton, khi người sử dụng clicks lên button thì nó Raise Event Click để cho ta xử lý trong Sub myCommandButton\_Click(), chẳng hạn. Class trong VB6 không có hỗ trợ Visual components, tức là không có chứa những controls như TextBox, Label ..v.v.. Tuy nhiên, ta có thể lấy những control có sẵn từ bên ngoài rồi chuyển cho Object của Class dùng.

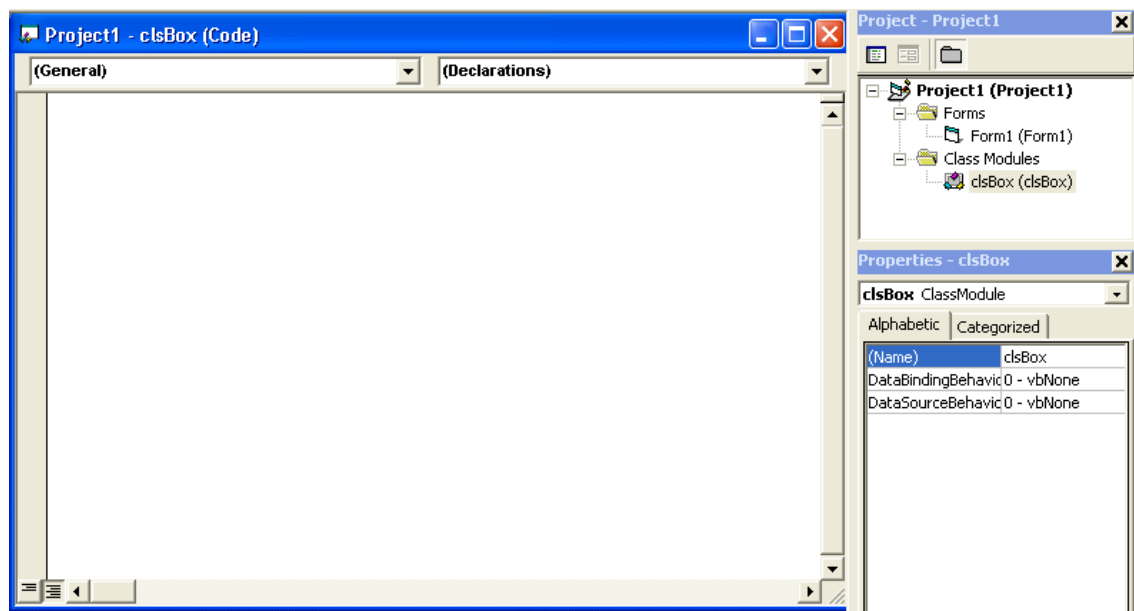
Bây giờ chúng ta hãy bắt đầu viết một Class. Chúng ta hãy mở một Project mới loại Standard EXE Visual Basic. Sau đó dùng Menu Command chọn **Add Class Module**:



Khi Add Class Module dialog hiện ra chọn **Class Module** và click Open.



Chúng ta sẽ thấy mở ra một khung trắng và Project Explorer với Properties Window. Trong Properties Window, hãy sửa thuộc tính Name của Class thành clsBox như dưới đây:



Kế đó đánh vào những dòng code dưới đây, trong đó có biểu diễn cách dùng Class clsBox.

```
Option Explicit
```

```

Private mX As Integer
Private mY As Integer
Private mWidth As Integer
Private mHeight As Integer

Public Property Let X(ByVal vValue As Integer)
    mX = vValue
End Property

Public Property Get X() As Integer
    X = mX
End Property

Public Property Let Y(ByVal vValue As Integer)
    mY = vValue
End Property

Public Property Get Y() As Integer
    Y = mY
End Property

Public Property Let Width(ByVal vValue As Integer)
    mWidth = vValue
End Property

Public Property Get Width() As Integer
    Width = mWidth
End Property

Public Property Let Height(ByVal vValue As Integer)
    mHeight = vValue
End Property

Public Property Get Height() As Integer
    Height = mHeight
End Property

Public Sub DrawBox(Canvas As Object)
    Canvas.Line (mX, mY)-(mX + mWidth, mY + mHeight), , B
End Sub

Public Sub ClearBox(Canvas As Object)
    Canvas.Line (mX, mY)-(mX + mWidth, mY + mHeight),
Canvas.BackColor, B
End Sub

```

Class clsBox có 4 Properties: X, Y, Width và Height. Ta sẽ dùng một ví dụ cụ thể là một hộp từ clsBox. Mỗi hộp có tọa độ (X,Y) và kích thước chiều rộng và chiều cao (width, height) của nó. Thật ra ta có thể dùng Public statement để khai báo các biến X, Y, Width và Height. Nhưng ở đây ta cố ý khai báo chúng là Private, dưới dạng mX, mY, mWidth và mHeight. Khi ta muốn thay đổi các trị số của chúng, ta sẽ dùng cùng một cách viết code như bình thường (ví dụ: myBox.X = 80 ). Nhưng khi chương trình xử lý assignment statement ấy, nó sẽ thực thi một loại phương thức (giống như Sub) gọi là thuộc tính **Let X (vValue)**. Ta thấy ở đây vValue

được gán cho mX (i.e.  $mX = vValue$ ), biến cục bộ của X. Như thế công việc này cũng chẳng khác gì sửa đổi một Public variable X. Tuy nhiên, ở đây ta có thể viết thêm code trong thuộc tính Let X để nó làm gì cũng được.

Mỗi lần chúng ta dùng cửa sổ thuộc tính để hiệu chỉnh kích thước chữ, màu chữ hay màu nền thì chẳng những các thuộc tính ấy của Label thay đổi, mà kết quả của sự thay đổi được có hiệu lực ngay lập tức, nghĩa là Label được hiển thị trở lại với trị số mới của thuộc tính. Đó là vì trong phương thức thuộc tính có cả mã lệnh bảo Label thực hiện việc hiển thị lại.

Ngược lại, khi ta dùng thuộc tính X của Object myBox, không phải ta chỉ đọc trị số thôi mà còn thực thi cả phương thức thuộc tính **Get X**. Nói tóm lại, thuộc tính cho ta cơ hội để thực thi một phương thức mỗi khi người sử dụng đọc hay viết trị số của biến ấy.

Ví dụ như nếu ta muốn kiểm soát để chỉ chấp nhận trị số tọa độ X mới khi nó không phải là số âm. Ta sẽ sửa thuộc tính Let X lại như sau:

```
Public Property Let X(ByVal vValue As Integer)
    If (vValue >= 0) Then
        mX = vValue
    End If
End Property
```

Thuộc tính có thể là **Read Only** hay **Write Only**. Nếu muốn một thuộc tính là Read Only thì ta không cung cấp thuộc tính Let. Nếu muốn một thuộc tính là Write Only thì ta không cung cấp thuộc tính Get. Ngoài ra nếu làm việc với **Object**, thay vì kiểu dữ liệu thông thường, thì ta phải dùng thuộc tính **Set**, thay vì thuộc tính Let.

Ví dụ ta cho clsBox một thuộc tính mới gọi là Font dùng đối tượng của lớp stdFont của VB6. Trong clsBox ta khai báo một biến cục bộ **mFont** và viết một thuộc tính **Set Font** như sau:

```
Private mFont As StdFont
Public Property Set Font(ByVal newFont As StdFont)
    Set mFont = newFont
End Property
```

Ta sẽ dùng thuộc tính Font của myBox (thuộc Class clsBox) như sau:

```
' Declare an object of Class StdFont of VB6
Dim myFont As StdFont
Set myFont = New StdFont
myFont.Name = "Arial"
myFont.Bold = True
Dim myBox As clsBox
Set myBox = New clsBox
Set myBox.Font = myFont ' Call the Property Set method
```

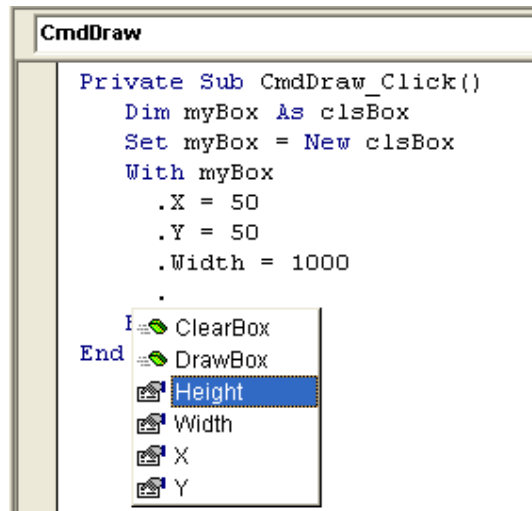
Class clsBox có hai Public Subs, **DrawBox** và **ClearBox**. ClearBox cũng vẽ một hộp như DrawBox, nhưng nó dùng BackColor của màn ảnh (canvas), nên coi như xóa cái hộp có sẵn. Do đó, nếu muốn, chúng ta có thể hiệu chỉnh Sub DrawBox để nhận một tùy chọn về màu sắc nét vẽ như sau:

```
Public Sub DrawBox(Canvas As Object, Optional fColor As Long)
    If IsMissing(fColor) Then
        Canvas.Line (mX, mY)-(mX + mWidth, mY + mHeight), , B
    Else
        Canvas.Line (mX, mY)-(mX + mWidth, mY + mHeight), fColor, B
    End If
End Sub
```

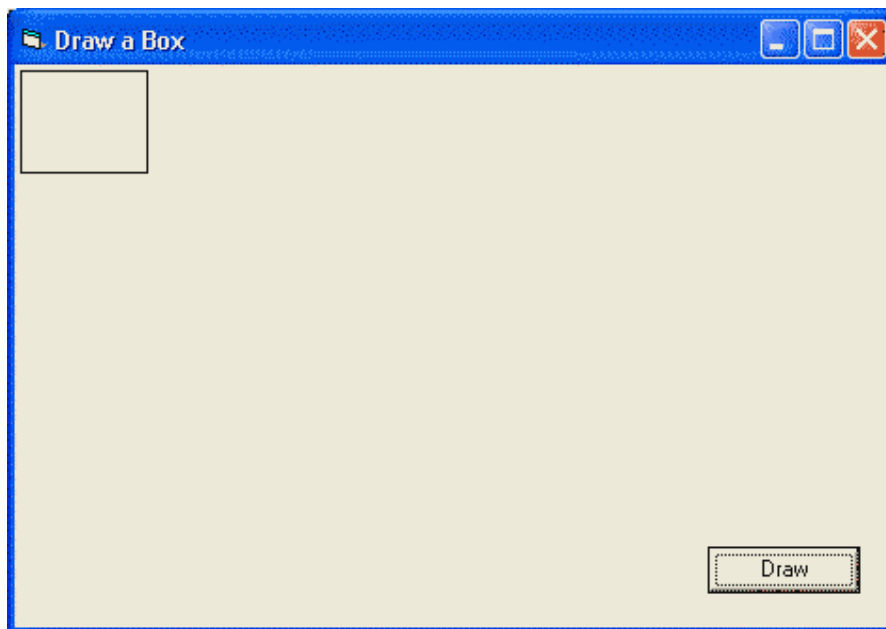
```
End If  
End Sub
```

Trong ví dụ trên, tham số tùy chọn `fColor` đã được kiểm tra bằng hàm **IsMissing**. Nếu `fColor` là `BackColor` của canvas thì ta sẽ có hiệu quả của `ClearBox`.

Trong form chính của chương trình dùng để test `clsBox`, mỗi khi ta tham chiếu đến một đối tượng thuộc lớp `clsBox`, IDE Intellisense sẽ hiển thị các Properties và Subs/Functions của `clsBox` như trong hình dưới đây:

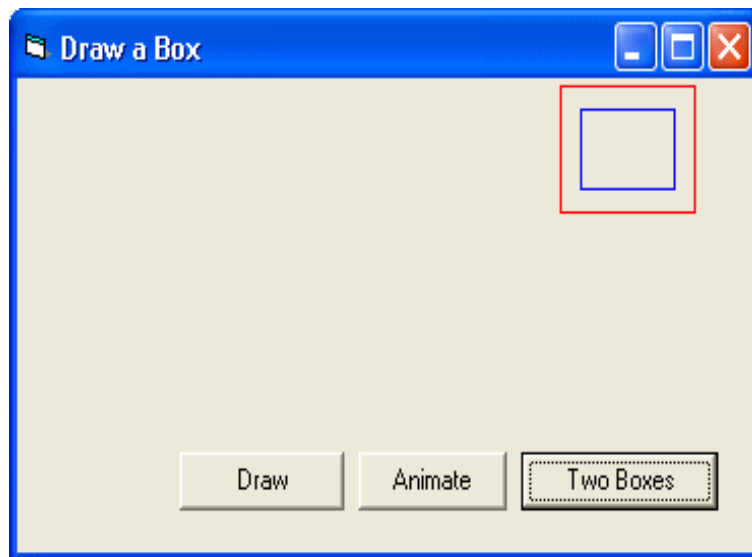


Trong chương trình này, mỗi khi ta click nút **Draw** thì một hộp được khởi tạo, cho tọa độ X,Y và kích thước Width, Height, rồi được vẽ ra ngay trên form.



Để cho chương trình sinh động hơn, khi người sử dụng clicks nút **Animate**, ta sẽ cho một hộp màu đỏ chạy từ trái qua phải.

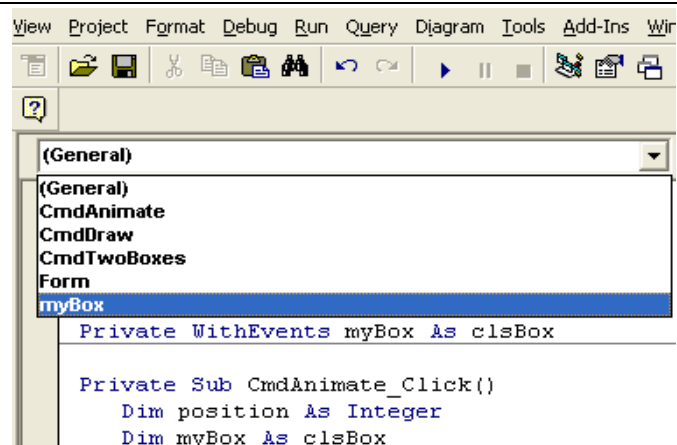
Khi người sử dụng clicks nút **Two Boxes** ta sẽ vẽ hai hộp, hộp trong màu xanh, hộp ngoài màu đỏ, và cho chúng chạy từ trái sang phải. Ở đây ta biểu diễn cho thấy mình muốn instantiate bao nhiêu hộp từ `clsBox` cũng được, và dĩ nhiên mỗi hộp có một bộ properties với giá trị riêng của nó.



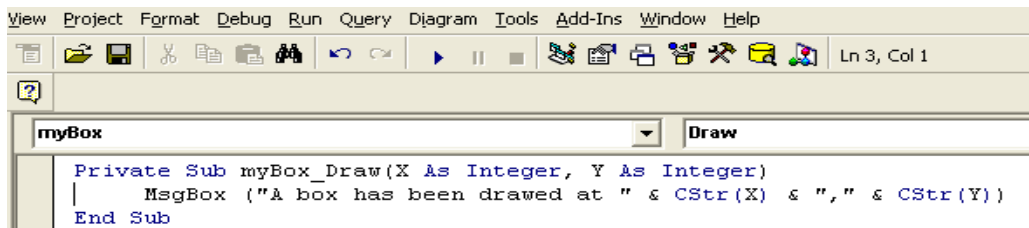
Ta có thể lập trình để cho Object báo cáo chương trình chủ của nó khi có một biến cố (Event) xảy ra bên trong Class.

Ta thử khai báo một Event tên Draw trong clsBox, và viết code để mỗi khi Sub DrawBox thực thi thì Class sẽ xây dựng một sự kiện Draw.

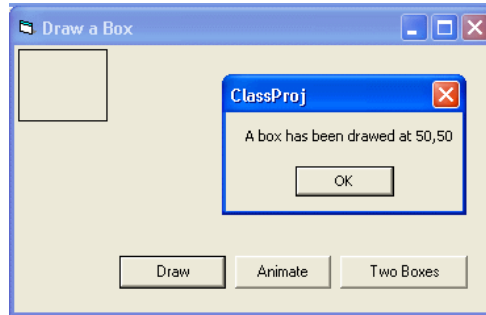
```
Public Event Draw(X As Integer, Y As Integer)
Public Sub DrawBox(Canvas As Object, Optional fColor As Long)
    If IsMissing(fColor) Then
        Canvas.Line (mX, mY)-(mX + mWidth, mY + mHeight), , B
    Else
        Canvas.Line (mX, mY)-(mX + mWidth, mY + mHeight), fColor, B
    End If
    RaiseEvent Draw(mX, mY)
End Sub
```



Bây giờ, trong frmClass thay vì chỉ khai báo Dim myBox as clsBox, ta sẽ khai báo Private WithEvents myBox as clsBox. Ngay sau đó, chữ myBox sẽ hiện ra trong danh sách các Object có hỗ trợ Event của frmClass. Kế đó ta sẽ viết chương trình để quản lý Event Draw của myBox, tức là ta cung cấp chương trình cho Private Sub myBox\_Draw (X as Integer, Y as Integer). Ở đây ta chỉ hiển thị một thông điệp báo cáo một hộp vừa được vẽ ở đâu.



Khi chạy chương trình, mỗi lần một đối tượng clsBox thực hiện Sub DrawBox ta sẽ thấy frmClass hiển thị một message giống như dưới đây.



Nhớ rằng, ta khai báo một Object với WithEvents khi ta muốn quản lý các Events của nó. Trong ví dụ trên, frmClass là chủ của myBox và nó nắm giữ Event Draw của myBox. Tương tự như vậy, ngay cả ở bên trong một Class, nếu Class ấy được giao cho một Object có thể Raise Events (ví dụ như TextBox, ListBox, Timer .v.v..), chúng ta cũng có thể khai báo Object ấy với các sự kiện kèm theo để nó có thể quản lý các Events của Object.

Trong ví dụ dưới đây ta viết các đoạn mã trong một Class đã được giao cho một TextBox khi form chính gọi Sub InitObject để chuyển cho Object một TextBox:

```
Private WithEvents mTextBox As TextBox
Public Sub InitObject(givenTextBox As TextBox)
    Set mTextBox = givenTextBox
End Sub
Private Sub mTextBox_KeyPress(KeyAscii As Integer)
    ' Place your code here to handle this event
End Sub
```



## CHƯƠNG 6. CÁC CHẾ ĐỘ HỘI THOẠI

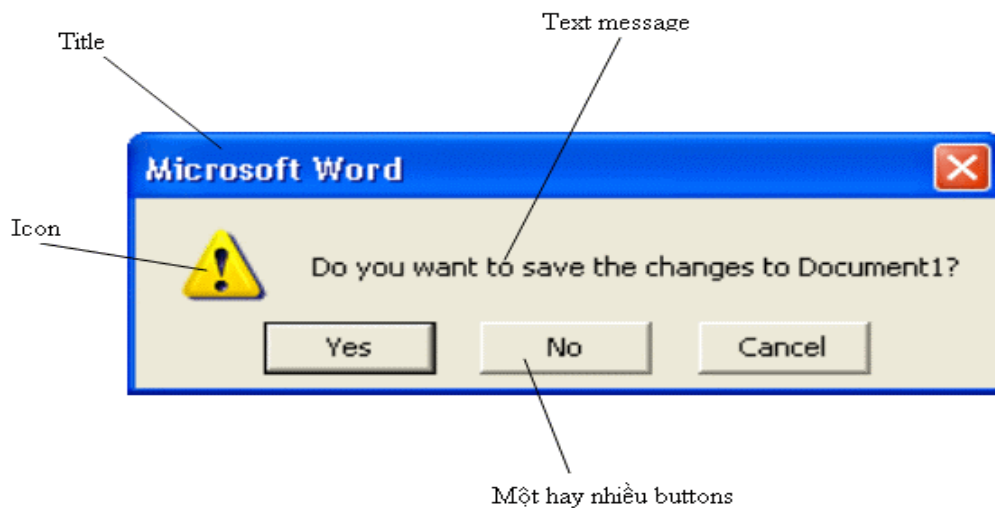
Dialogs (hội thoại) được dùng để hiển thị tin tức và nhận thông tin từ chuột hay bàn phím từ người sử dụng tùy theo tình huống. Chúng được dùng để tập trung sự chú ý của người sử dụng vào công việc hiện tại của chương trình nên rất hữu dụng trong các chương trình của Windows.

Có nhiều dạng Dialogs, mỗi dạng được áp dụng cho một ngữ cảnh riêng biệt. Trong chương này ta sẽ bàn qua 4 loại dialogs chính và xem xét khi nào và bằng cách nào ta dùng chúng:

- Message Boxes
- Input Boxes
- Common Dialogs
- Custom Dialogs

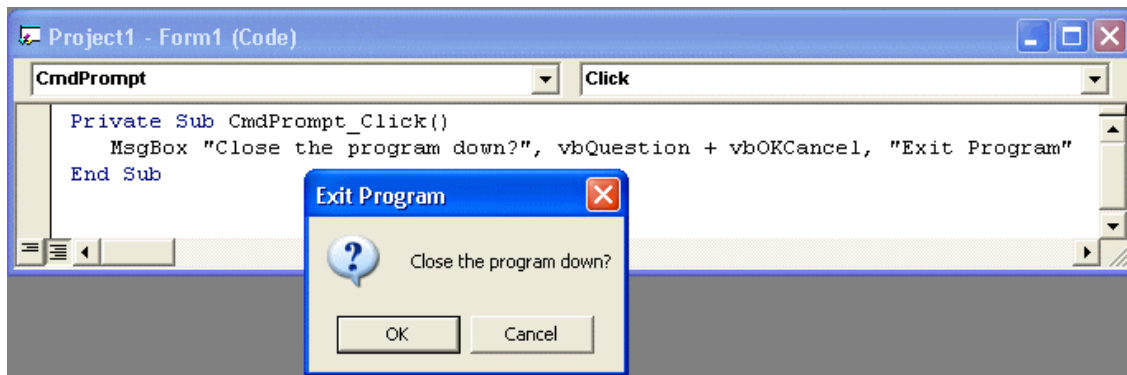
### I. Message Boxes (hộp thông điệp)

Message Boxes được dùng để nhắc nhở người sử dụng một chuyện gì, và đòi hỏi một phản ứng nào đó từ người sử dụng. Ví dụ, khi ta chấm dứt chương trình MSWord mà chưa lưu trữ hồ sơ thì MSWord sẽ nhắc ta lưu trữ nó bằng Dialog dưới đây:



Trong trường hợp này người sử dụng có thể click một trong 3 buttons. Nếu click **Yes** thì sẽ xúc tiến việc lưu trữ hồ sơ trước khi kết thúc chương trình MSWord. Nếu click **No** thì MSWord sẽ lặng lẽ kết thúc mà không lưu trữ hồ sơ. Nếu click **Cancel** thì có nghĩa người sử dụng đổi ý việc chấm dứt chương trình và trở lại tiếp tục dùng MSWord.

Ta dùng **MsgBox** để hiển thị Message Box như trong hình dưới đây:



Tham số thứ nhất của MsgBox là thông điệp “**Close the program down?**”, tham số thứ hai là tập hợp của icon (vbQuestion) và số buttons (vbOKCancel) bằng cách cộng hai hằng: **vbQuestion + vbOKCancel** (hai buttons OK và Cancel), tham số thứ ba là title (tiêu đề) của Dialog.

Trong ví dụ MSWord trên hằng số của icon và buttons là **vbExclamation + vbYesNoCancel** (ba nút Yes, No và Cancel).

Ta chọn số và loại buttons theo bảng dưới đây:

Tên hằng	Các buttons
vbOKOnly	OK
vbOKCancel	OK Cancel
vbYesNo	Yes No
vbRetryCancel	Retry Cancel
vbYesNoCancel	Yes No Cancel
vbAbortRetryIgnore	Abort Retry Ignore

Hằng số của các icons ta có thể dùng là **vbCritical**, **vbQuestion**, **vbExclamation** và **vbInformation**.

Khi một Message Box được mở ra, cả chương trình ngừng lại và đợi người sử dụng phản ứng. Ta nói Message Box được hiển thị trong **Modal Mode**, nó dành mọi sự chú ý và tạm ngưng các sự thực thi khác trong cùng chương trình. Sau khi người sử dụng click một button, Message Box sẽ biến mất và chương trình sẽ tiếp tục chạy từ dòng code ngay dưới dòng MsgBox.

Trong ví dụ trên ta dùng MsgBox như một Sub, nhưng ta cũng có thể dùng MsgBox như một Function để biết người sử dụng vừa mới click button nào. Function MsgBox returns một value (trả về một giá trị) mà ta có thể thử để theo đó thì hành. Ví dụ như:

```
Private Sub CmdPrompt_Click()
    Dim ReturnValue As Integer
    ReturnValue = MsgBox("Close the program down", vbQuestion +
vbOKCancel, "Exit Program")
    Select Case ReturnValue
    Case vbOK
        MsgBox "You clicked OK"
    Case vbCancel
        MsgBox "You clicked Cancel"
```

```
End Select
End Sub
```

Các trị số Visual Basic intrinsic constants mà Function MsgBox returns là:

Trị số	Tên nút	Tên hằng
1	OK	vbOK
2	Cancel	vbCancel
3	Abort	vbAbort
4	Retry	vbRetry
5	Ignore	vbIgnore
6	Yes	vbYes
7	No	vbNo

Chúng ta có thể hiển thị thông điệp trong Message Box thành nhiều dòng bằng cách dùng hằng **vbCrLf** (CarriageReturn và LineFeed) để đánh dấu những chỗ ngắt khúc như sau:

```
MsgBox "This is the first line" & vbCrLf & " followed by the  
second line"
```

Nếu chúng ta thấy mình thường dùng MsgBox với cùng một icon và những buttons, nhưng có thông điệp khác nhau, chúng ta có thể viết một Global Subroutine trong mô-đun .BAS để dùng lại nhiều lần. Ví dụ chúng ta có một Global Sub như sau:

```
Public Sub DisplayError(ByVal ErrMess As String )
    MsgBox ErrMess, vbCritical + vbOKOnly, "Error"
End Sub
```

Mỗi lần muốn hiển thị một Error message chúng ta chỉ cần gọi Sub DisplayError với nội dung thông điệp mà không sợ dùng nhầm lẫn icon. Sau này muốn đổi cách hiển thị thông điệp Error chỉ cần edit ở một chỗ. Nếu người sử dụng muốn chúng ta lưu trữ tất cả mọi errors xảy ra lúc run-time, chúng ta chỉ cần thêm vài dòng code trong Sub DisplayError để viết thông báo lỗi vào một tập tin dạng Text.

## II. Input Boxes

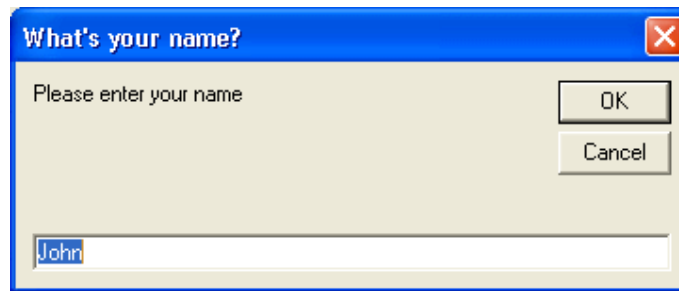
Với Message Boxes, người sử dụng chỉ có thể click lên một button. Trong thực tế, đôi khi ta muốn người sử dụng đánh vào thêm một ít dữ kiện, trong trường hợp ấy ta có thể dùng **Input Boxes**.

Input Boxes giống giống Message Box, nhưng nó chuyên nhận dữ liệu đầu vào từ người sử dụng và không hiển thị một biểu tượng. Ví dụ:

```
Private Sub CmdGreeting_Click()
    Dim strReply As String
    strReply = InputBox$("Please enter your name", "What 's your  
name?", "John", 2000, 1000)
    MsgBox "Hi " & strReply & ", it 's great to meet you!",  
vbOKOnly, "Hello"
End Sub
```

Để ý các tham số của **Function InputBox\$**. Tham số thứ nhất là Text Message, tham số thứ hai là Title của Dialog, tham số thứ ba là Default Input Value. Đây là giá trị được hiển thị

sẵn trong Input Box khi nó xuất hiện, nếu đó là đầu vào mà người sử dụng thường đánh vào thì người sử dụng chỉ cần click nút **OK** là đủ. Hai tham số cuối cùng là Optional (tùy chọn, có cũng được, không có cũng không sao). Nó là tọa độ X,Y của Input Box trong đơn vị **twips**. Hệ thống tọa độ lấy góc trên bên trái làm chuẩn với X=0, Y=0.



Input Box có hai dạng hàm:

- **InputBox\$** - trả về một chuỗi dạng chuỗi
- **InputBox** - trả về một chuỗi nằm trong biến Variant

Nếu chúng ta click nút Cancel thì giá trị trả về là chuỗi rỗng (empty string), chúng ta có thể thử chuỗi rỗng để nhận diện trường hợp này.

Dưới đây là một ví dụ dùng hàm InputBox:

```
Private Sub CmdFortuneTeller_Click()  
    Dim varValue As Variant  
    Dim intAge As Integer  
    varValue = InputBox("Please enter your age", "How old are  
you?", "18")  
    If IsNumeric(varValue) Then  
        intAge = Val(varValue)  
        If intAge < 20 Then  
            MsgBox "You are a young and ambitious person", vbOKOnly,  
"Observation"  
        Else  
            MsgBox "You are a matured and wise person", vbOKOnly,  
"Observation"  
        End If  
    Else  
        MsgBox "Oh oh! - please type your age!", vbCritical +  
vbOKOnly, "Input Error"  
    End If  
End Sub
```

Mặc dầu Input Boxes rất dễ dùng, trên thực tế rất ít khi ta dùng nó vì những lý do sau đây:

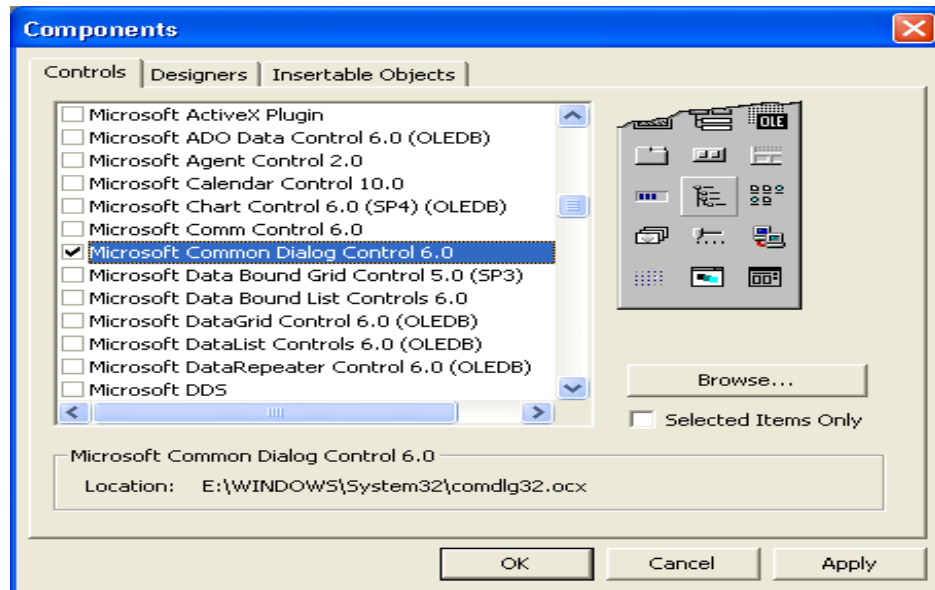
- Ta không thể làm gì được trong lúc người sử dụng nhập dữ liệu, phải đợi sau khi người sử dụng click OK thì mới bắt đầu xử lý chuỗi văn bản đầu vào. Ngược lại nếu ta dùng một Textbox trong một Form thông thường, ta có thể code trong các bộ quản lý sự kiện của Events **KeyPress** hay **Change** để kiểm soát các sự bấm phím (keystrokes) của người sử dụng.
- Input Boxes chỉ cho ta đánh vào một chuỗi văn bản duy nhất. Nhiều khi ta muốn người sử dụng đánh vào nhiều thứ nên cần phải có một form riêng.

- Sau cùng, Input Boxes xem không đẹp mắt. Chương trình dùng Input Boxes có vẻ như không chuyên nghiệp, do đó ta cần phải dùng **Custom Dialogs**.

### III. Common Dialogs

Chúng ta có thể thấy hầu như mọi chương trình trong Windows đều có cùng những dialogs để mở và ghi tập tin ? Và hầu như tất cả chương trình đều có cùng dialogs để chọn màu, phong chữ hay để in ? Đó là vì các Dialogs thông dụng ấy thuộc về Common Dialog Library của MSWindows và cho phép các chương trình gọi.

Muốn dùng các Dialogs ấy trong VB6 ta phải reference **Comdlg32.ocx** bằng IDE Menu command **Project | Components...** rồi chọn và Apply **Microsoft Common Dialog Control 6.0**.



Microsoft Common Dialog Control 6.0 cho ta sáu dạng Dialogs tùy theo gọi Method nào:

Tên	Method
Open File	ShowOpen
Save File	ShowSave
Color	ShowColor
Font	ShowFont
Print	ShowPrinter
Help	ShowHelp

### IV. Open và Save File Dialogs

Chúng ta hãy mở một Project mới với một button tên CmdOpen trong Form1 và đánh vào code sau đây cho Sub CmdOpen\_Click:

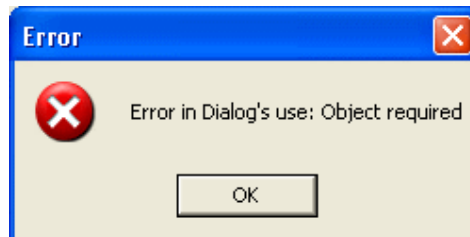
```
Private Sub CmdOpen_Click()
    On Error GoTo DialogError
    With CommonDialog1
```

```

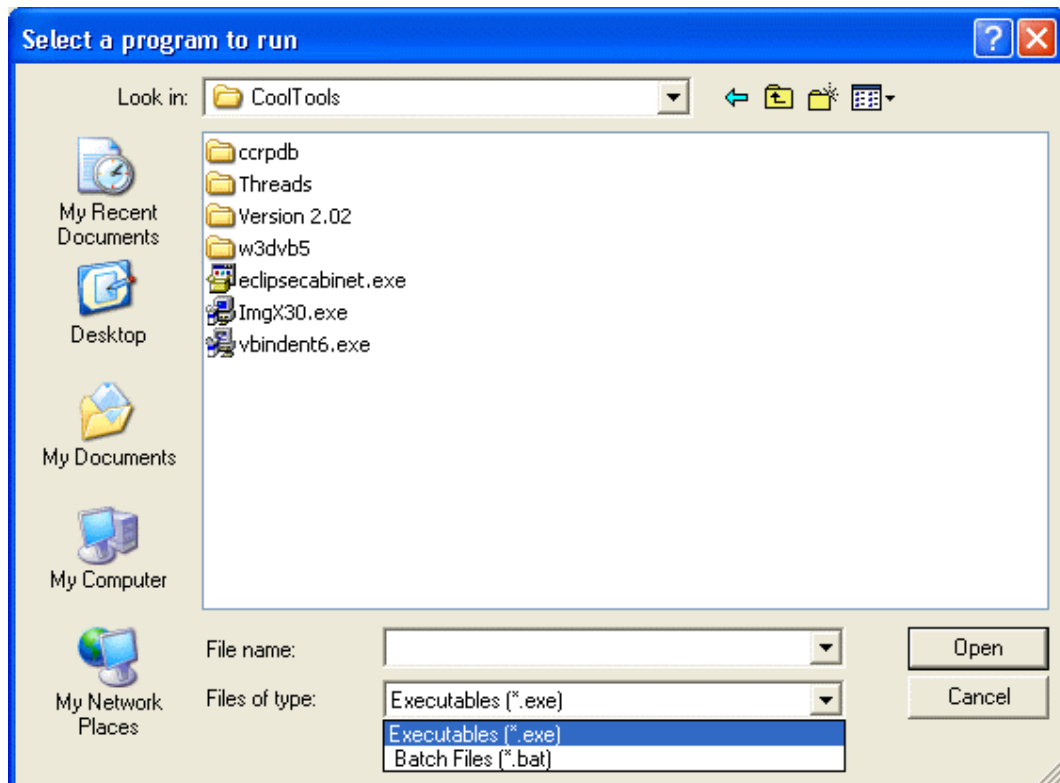
        .CancelError = True ' Generate Error number cdlCancel if
user click Cancel
        .InitDir = "E:\VB6" ' Initial (i.e. default ) Folder
        .Filter = "Executables (*.exe) | *.exe| Batch Files
(*.bat) | *.bat"
        .FilterIndex = 1 ' Select ""Executables (*.exe) | *.exe"
as default
        .DialogTitle = "Select a program to run"
        .ShowOpen ' Launch the Open Dialog
        MsgBox "You selected " & .FileName, vbOKOnly +
vbInformation, "Open Dialog"
        End With
        Exit Sub
    DialogError:
        If Err.Number = cdlCancel Then
            MsgBox "You clicked Cancel!", vbOKOnly + vbInformation,
"Open Dialog"
            Exit Sub
        Else
            MsgBox "Error in Dialog's use: " & Err.Description,
vbOKOnly + vbCritical, "Error"
            Exit Sub
        End If
    End Sub
End Sub

```

Hãy chạy chương trình ấy và click trên nút **Open**, chương trình sẽ hiển thị thông báo lỗi dưới đây:



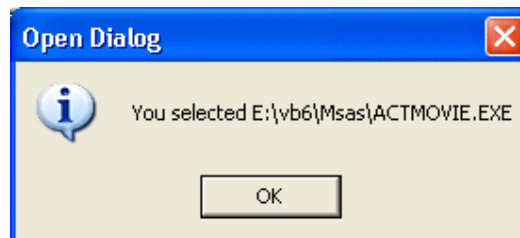
Lý do là ta quên bỏ một Microsoft Common Dialog Control 6.0 vào Form1. Vậy chúng ta hãy nhấn đúp chuột tại biểu tượng của nó trong ToolBox. Bây giờ hãy chạy chương trình lại và click button Open để hiển thị **Open Dialog**.



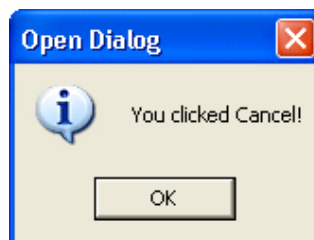
Chúng ta có thể chọn thư mục nào tùy ý bằng cách di chuyển từ thư mục này qua thư mục khác hay thay đổi ổ đĩa. Nếu chúng ta click vào bên phải của combobox **File of type**, nó sẽ dropdown để cho thấy chúng ta có thể chọn một trong hai loại Files như liệt kê trong câu lệnh:

```
.Filter = "Executables (*.exe) | *.exe| Batch Files (*.bat) | *.bat"
```

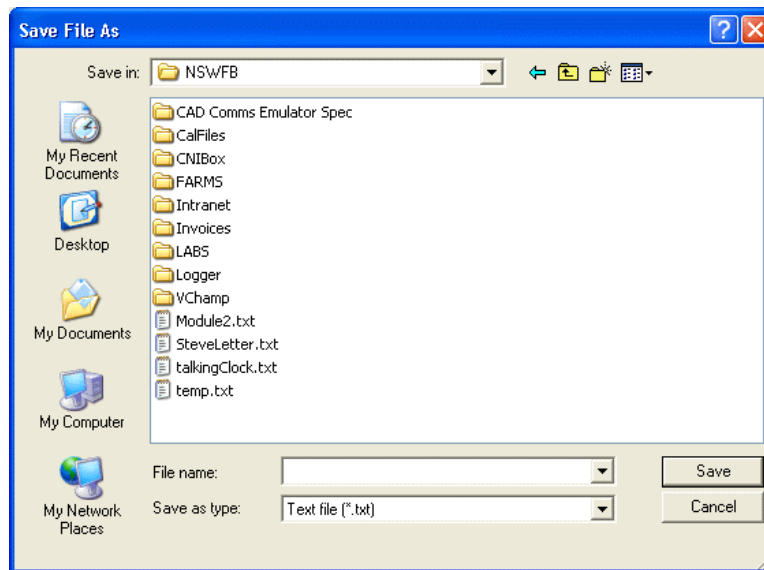
Sau khi chọn một Filename có sẵn hay đánh một tên vào **File name** textbox, chúng ta click **Open**. Sau đó, `CommonDialog1.FileName` sẽ chứa tên file chúng ta đã chọn hay đánh vào.



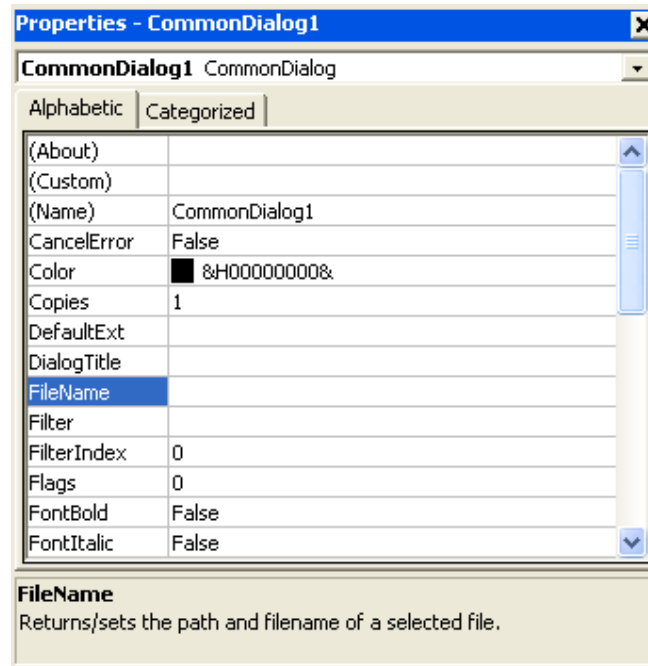
Vì ta cho `.CancelError = True` nên nếu người sử dụng click Cancel chương trình sẽ sinh ra một lỗi với mã số **32755 (cdlCancel)**. Ở đây ta bắt Error ấy bằng cách dùng **On Error GoTo DialogError** và thử `Err.Number= cdlCancel` để hiển thị Error message dưới đây:



**Save Dialog** cũng tương tự như Open Dialog, ta dùng method **ShowSave** để hiển thị nó.

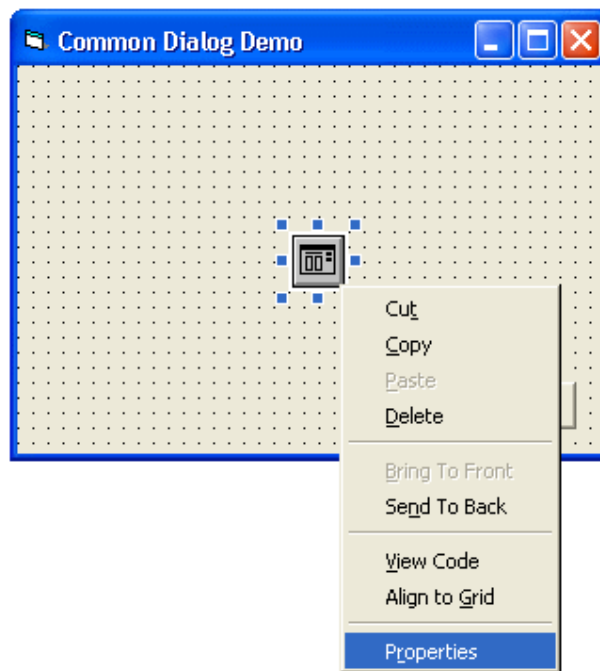


Trong ví dụ trên ta định nghĩa các thuộc tính của CommonDialog1 bằng code. Chúng ta cũng có thể dùng Thuộc tính Windows để định nghĩa chúng như dưới đây:

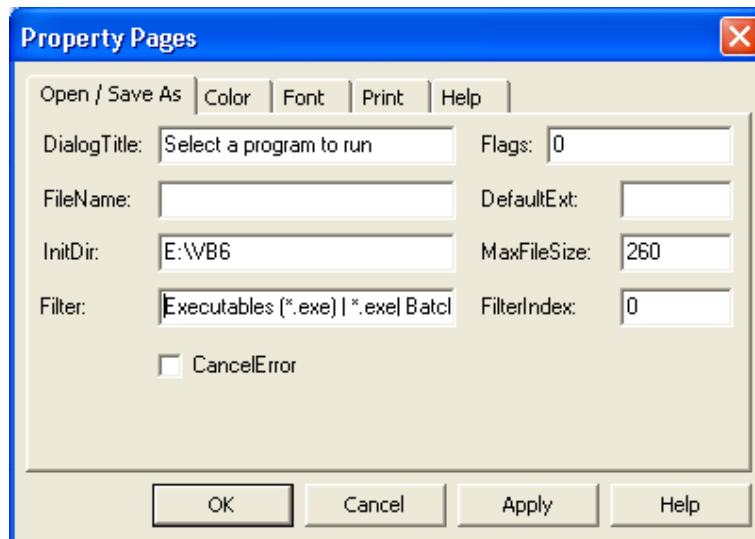


Ngoài ra, chúng ta cũng có thể dùng các trang thuộc tính của CommonDialog1 để định nghĩa thuộc tính lúc thiết kế bằng cách right click CommonDialog1 trên Form1 rồi chọn thuộc tính:





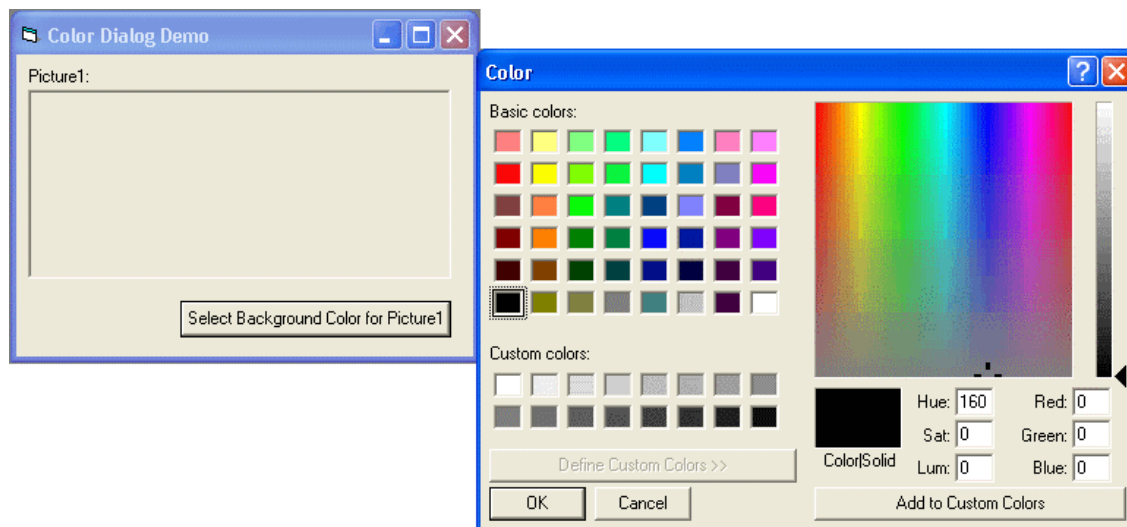
Thuộc tính Pages Dialog sẽ hiển thị với Tab **Open/Save As** có sẵn lúc đầu, chúng ta có thể đánh các tin tức như sau:



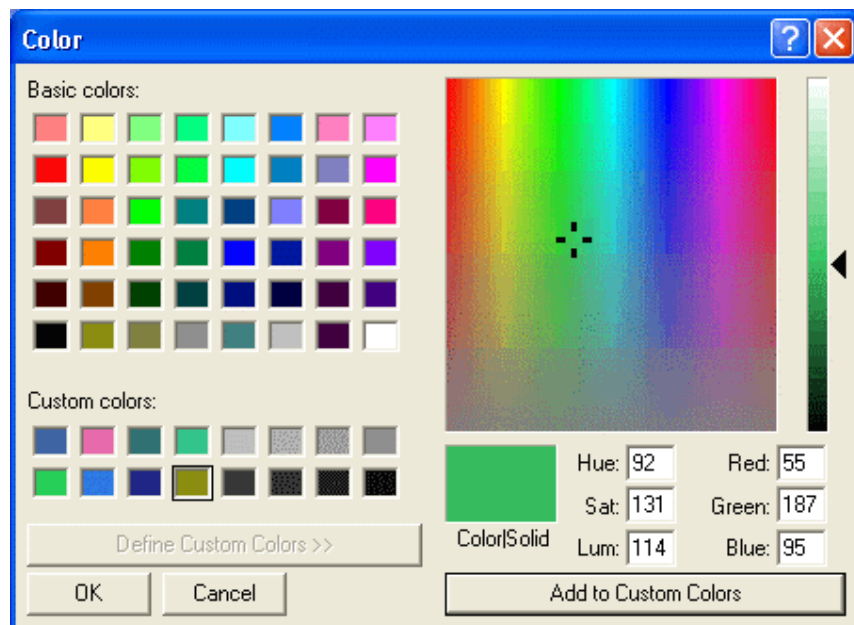
## V. Các loại Dialog có sẵn để dùng

### 1. Color Dialog

**Color Dialog** cho người sử dụng một cách chọn màu rất dễ dùng. Ngoài những màu có sẵn, người sử dụng có thể tự tạo ra một màu rồi cho nó thêm vào trong bảng màu được cung cấp, gọi là **Windows Palette** bằng cách click button **Add to Custom Colors**.



Chúng ta tạo ra một màu bằng cách click chỗ có màu theo ý trong bảng màu lớn hình vuông rồi nắm hình tam giác bên phải kéo lên, kéo xuống để thay đổi độ đậm của màu như hiển thị trong hộp vuông **Color|Solid**. Khi vừa ý với màu hiển thị, chúng ta click button **Add to Custom Colors**, màu ấy sẽ được cho thêm vào nhóm **Custom Colors** nằm phía dưới, bên trái.



Ta dùng method **ShowColor** để hiển thị Color Dialog. Sau khi người sử dụng đã chọn một màu rồi, ta có thể trực tiếp assign nó cho property ForeColor hay BackColor của một control. Trong ví dụ dưới đây cái màu mà người sử dụng vừa chọn được assigned cho background của picturebox Picture1:

```
Private Sub CmdSelectColor_Click()
    On Error GoTo NoColorChosen
    With CommonDialog1
        .CancelError = True
        ' Entire dialog box is hiển thị, including the Define
        Custom Colors section
        .Flags = cdlCCFullOpen
        .ShowColor ' Launch the Color Dialog
    End With
End Sub
```

```

        Picture1.BackColor = .Color ' Assign selected color to
background of Picture1
    Exit Sub
End With
NoColorChosen:
    ' Get here if user clicks the Cancel button
    MsgBox "You did not select a color!", vbInformation,
"Cancelled"
    Exit Sub
End Sub

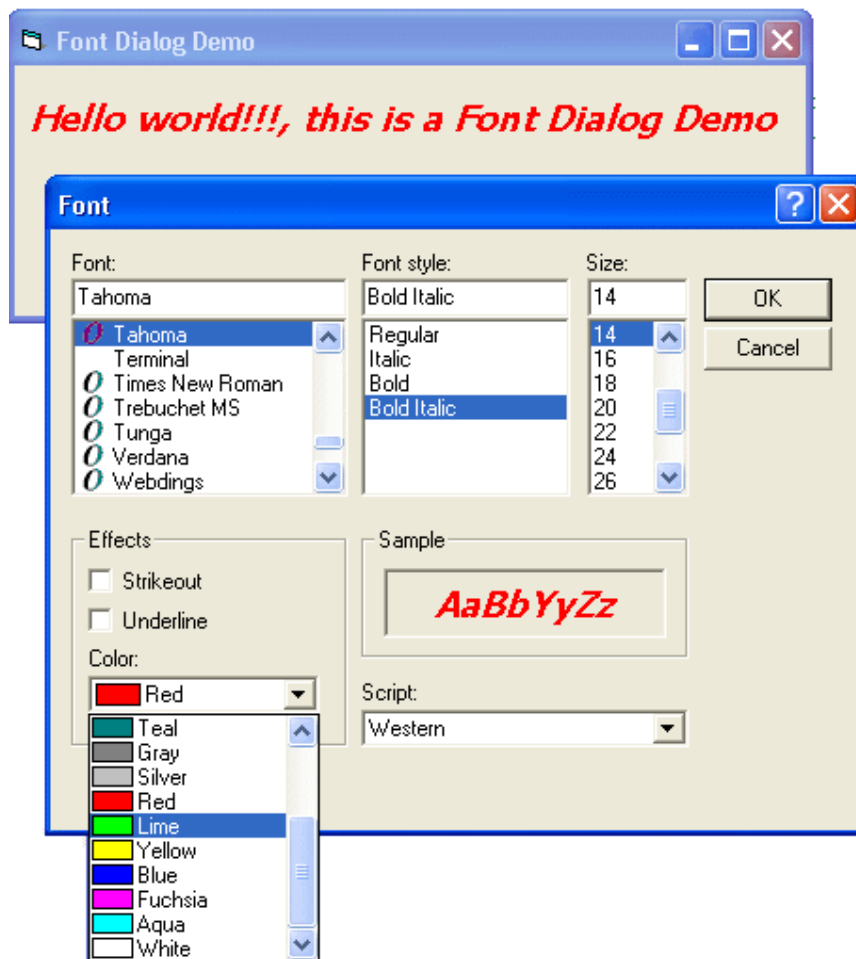
```

## 2. Font Dialog

**Font Dialog** cho ta chọn Font cho màn ảnh hay printer và chọn màu để dùng cho chữ của Font. Ta dùng method **ShowFont** để hiển thị FontDialog. Các chi tiết trình bày trong Font Dialog tùy thuộc vào trị số của Flags như sau:

Tên hằng	Trị số	Hiệu quả
cdlCFScreenFonts	1	Chỉ hiển thị các Fonts printer hỗ trợ
cdlCFPrinterFonts	2	Chỉ hiển thị các Fonts của màn ảnh, chưa chắc tất cả đều được printer hỗ trợ
cdlCFBoth	3	Hiển thị các Fonts màn ảnh và printer
cdlCFScalableOnly	&H20000	Chỉ hiển thị các scalable Fonts như TrueType fonts mà chúng ta đã cài vào máy

Nếu chúng ta muốn cho người sử dụng tùy chọn để chọn màu thì thêm 256 vào trị số của Flags.



Dưới đây là code để cho người sử dụng chọn Font và màu của Label1.

```
Private Sub CmdSelectFont_Click()
    On Error GoTo NoFontChosen
    CommonDialog1.CancelError = True
    ' Causes the dialog box to list only the screen fonts
    supported by the system.
    CommonDialog1.Flags = cdlCFScreenFonts + 256 ' Add 256 to
include Color option
    CommonDialog1.ShowFont ' Launch the Font Dialog
    With Label1.Font
        .Bold = CommonDialog1.FontBold
        .Italic = CommonDialog1.FontItalic
        .Name = CommonDialog1.FontName
        .Size = CommonDialog1.FontSize
        .Strikethrough = CommonDialog1.FontStrikethru
        .Underline = CommonDialog1.FontUnderline
    End With
    Label1.ForeColor = CommonDialog1.Color
    Label1.Caption = "Hello world!!!, this is a Font Dialog Demo"
    Exit Sub
NoFontChosen:
    MsgBox "No font was chosen!", vbInformation, "Cancelled"
    Exit Sub
End Sub
```

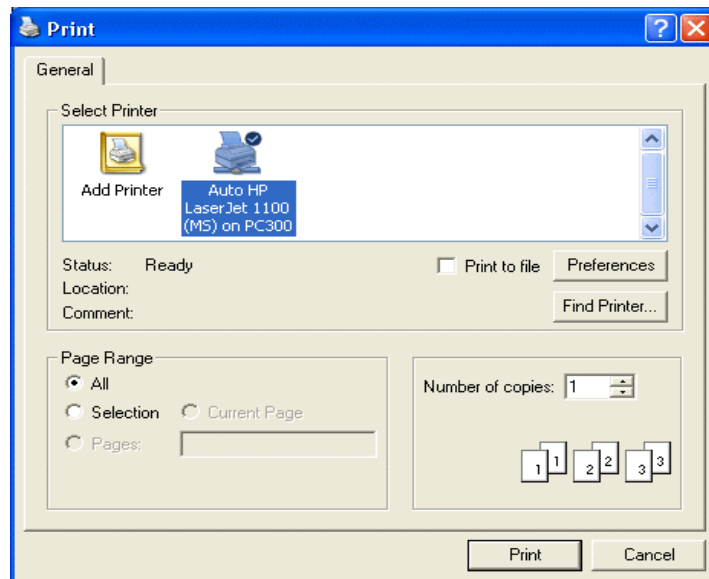
Chú ý: Nếu chúng ta quên cho Flags một trong những hằng số nói trên chương trình sẽ cho một Error message như sau:



### 3. Print Dialog

**Print Font** cho ta một giao diện cũng giống như trong Microsoft Office để chọn những tùy chọn về việc in. Với Print Dialog ta có thể chọn printer nào với những đặc tính nào bằng cách nhấn nút **Properties** hay nút **Preferences**. Ta cũng có thể quyết định in từ trang nào đến trang nào của tài liệu và in bao nhiêu bản sao. Chỉ có điều phải lưu ý là nếu người sử dụng dùng Print Dialog để chọn một Printer khác mà trong Print Dialog ta đã chọn Property **PrinterDefault = True** thì Printer ấy sẽ trở thành Default Printer và nó cũng sẽ có hiệu lực vĩnh viễn trong cả Windows cho đến khi người sử dụng thay đổi lại.

Khác với Color và Font Dialogs, Print Dialog không đòi hỏi ta phải cho một trị số của Property Flags. Ta chỉ cần dùng Method **ShowPrinter** để hiển thị Print Dialog. Ba thuộc tính thường được dùng nhất sau khi người sử dụng chọn các tùy chọn của Print Dialog là **Copies**, **FromPage** và **ToPage**. Để cho người sử dụng các giá trị mặc định của những thuộc tính này, chúng ta có thể để sẵn các trị số trước khi hiển thị Print Dialog.



Dưới đây là code mẫu dùng print Dialog:

```
Private Sub CmdSelectPrinter_Click()  
    With CommonDialog1  
        .FromPage = 1  
        .ToPage = 1  
        .Copies = 1  
        .ShowPrinter  
    End With  
End Sub
```

#### 4. Help Dialog

Ta dùng method **ShowHelp** để hiển thị các thông tin giúp đỡ, nhưng nhớ phải cho **CommonDialog1** ít nhất trị số của các thuộc tính **HelpFile** và **HelpCommand**.

```
Private Sub CmdHelp_Click()  
    CommonDialog1.HelpFile = "YourProgram.hlp"  
    CommonDialog1.HelpCommand = cdlHelpContents  
    CommonDialog1.ShowHelp  
End Sub
```

Để biết thêm chi tiết về cách dùng **ShowHelp**, highlight chữ **HelpContext** trong source code VB6 rồi ấn phím **F1** và chọn **MsComDlg**.

#### VI. Custom Dialogs

Nhiều khi Message Box, Input Box hay các dạng Common Dialogs vẫn không thích hợp cho hoàn cảnh lập trình. Trong trường hợp ấy chúng ta có thể dùng một Form bình thường để làm thành một Dialog theo yêu cầu. Nó hơi mất công hơn một chút, nhưng thứ nhất nó có những màu sắc giống như các Forms khác trong chương trình, và thứ hai ta muốn làm gì tùy ý. Chỉ có cái bất lợi là chương trình sẽ dùng nhiều tài nguyên hơn và cần thêm một ít bộ nhớ.

Sau đây ta thử triển khai một Login Form tổng quát, có thể dùng trong nhiều trường hợp. Khi khởi động, chương trình này sẽ hiển thị một Login form yêu cầu người sử dụng đánh vào tên và mật khẩu. Sau đó, nếu tên và mật khẩu hợp lệ thì cái Form chính của chương trình mới hiện ra. Cách ta thực hiện là cho chương trình khởi động với một **Sub Main** trong .BAS Module. Sub Main sẽ gọi **Sub GetUserInfo** (cũng nằm trong cùng Module) để hiển thị form frmLogin trong Modal mode để nó làm việc cùng một cách như Message Box, Input Box hay Common Dialogs.

Khi form frmLogin được dấu kín bằng statement **Me.Hide** thì execution trong Sub **GetUserInfo** sẽ tiếp tục để chi tiết điền vào các textboxes txtUserName và txtPassword được trả về local variables strUserName và strPassword. Mã nguồn của Sub Main và Sub **GetUserInfo** được liệt ra dưới đây:

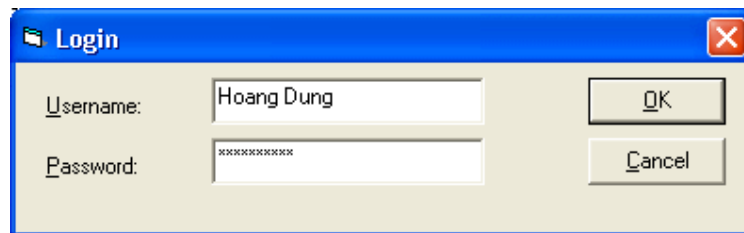
```
Sub Main()  
    Dim strUserName As String  
    Dim strPassword As String  
    ' Call local Sub getUserInfo to obtain UserName and Password  
    GetUserInfo strUserName, strPassword  
    If strUserName = "" Then  
        MsgBox "Login failed or aborted", vbInformation, "login  
Aborted"  
    Else  
        MsgBox "User " & strUserName & " logged in with password "  
& strPassword, vbInformation, "Login accepted"  
        ' Check UserName and Password here  
        ' If valid password then show the Main form of the program  
        ' which is implemented separately...  
        ' frmMain.Show  
    End If  
End Sub
```

```

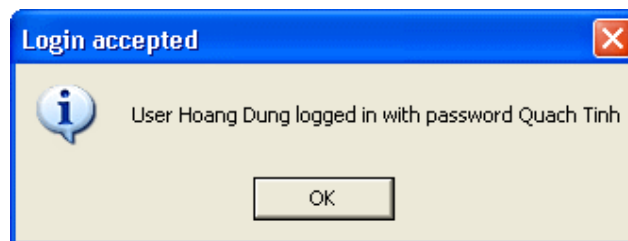
Private Sub GetUserInfo(ByRef sUserName As String, ByRef
sPassword As String)
    ' Invoke frmLogin form in Modal mode
    frmLogin.Show vbModal
    ' As soon as frmLogin is hidden, the execution gets here
    sUserName = frmLogin.txtUserName ' assign the form's
txtUserName to sUserName
    sPassword = frmLogin.txtPassword ' assign the form's
txtPassword to sPassword
    Unload frmLogin ' Unload form frmLogin
End Sub

```

Login form được hiển thị như dưới đây:



Sau khi user điền chi tiết và click **OK**, tạm thời ta chỉ hiển thị một thông điệp để xác nhận các chi tiết ấy.



Trong tương lai, chúng ta có thể viết thêm code để kiểm tra xem tên và mật khẩu có hiệu lực không. Có một vài chi tiết về form frmLogin để nó làm việc giống một Common Dialog:

Ta cho property **BorderStyle** của frmLogin là **Fixed Dialog**.

Ta cho **Property PasswordChar** của textbox txtPassword bằng "\*" để khi người sử dụng điền mật khẩu, ta chỉ thấy một dòng dấu hoa thị.

Ta cho Property **StartPosition** của form là **CenterScreen**.

**Property Default** của button cmdOK là **True** để khi người sử dụng ấn phím **Enter** trong form là coi như tương đương với click button cmdOK.

Tương tự như thế, **Property Cancel** của button cmdCancel là **True** để khi người sử dụng ấn phím **Esc** trong form là coi như tương đương với click button cmdCancel.

Tạm thời coding của event click của cmdOK và cmdCancel chỉ đơn giản như liệt kê dưới đây:

```

Sub Main()
    Dim strUserName As String
    Dim strPassword As String
    ' Call local Sub getUserInfo to obtain UserName and Password
    GetUserInfo strUserName, strPassword
    If strUserName = "" Then

```

```

        MsgBox "Login failed or aborted", vbInformation, "login
Aborted"
    Else
        MsgBox "User " & strUserName & " logged in with password "
& strPassword, vbInformation, "Login accepted"
        ' Check UserName and Password here
        ' If valid password then show the Main form of the program
which is implemented separately...
        ' frmMain.Show
    End If
End Sub

Private Sub GetUserInfo(ByRef sUserName As String, ByRef
sPassword As String)
    ' Invoke frmLogin form in Modal mode
    frmLogin.Show vbModal
    ' As soon as frmLogin is hidden, the execution gets here
    sUserName = frmLogin.txtUserName ' assign the form's
txtUserName to sUserName
    sPassword = frmLogin.txtPassword ' assign the form's
txtPassword to sPassword
    Unload frmLogin ' Unload form frmLogin
End Sub

```



## CHƯƠNG 7. THIẾT KẾ ĐỒ HOẠ

Visual Basic 6 có cho ta một số công cụ về đồ họa (graphics) để trang điểm cho các cửa sổ thêm phong phú, thân thiện, dễ làm việc và đẹp mắt hơn. Dù các phương tiện về đồ họa này không đủ mạnh để ta viết những chương trình đòi hỏi kỹ thuật đồ họa cao như trò chơi (games), thiết kế các bản vẽ kỹ thuật (CAD) nhưng có thể đáp ứng các yêu cầu cơ bản về đồ họa khi thiết kế giao diện cho các phần mềm ứng dụng trong quản lý, hệ thống.

Khi nói đến đồ họa, ta muốn phân biệt nó với chế độ văn bản (text) thông thường. Ví dụ ta dùng Notepad để soạn thảo một bài thơ, trong lúc bài thơ đang được hiển thị ta có thể sửa đổi dễ dàng bằng cách dùng bàn phím để đánh thêm các chữ mới vào, dùng các nút Delete, Backspace để xóa các chữ. Đó là ta làm việc với chế độ Text.

Tiếp đến, trong khi bài thơ còn đang hiển thị, ta dùng một chương trình Graphic để bổ sung thêm các hình ảnh minh họa hoặc chuyển văn bản bài thơ thành file ảnh thì ta có một Graphic. Sau đó, muốn sửa đổi bài thơ từ graphic này ta phải dùng một graphic editor như MSPaint, PhotoShop, PaintShopPro...

### I. Màu (color) và độ phân giải (resolution)

Ta nói một tấm hình tốt vì nó có màu sắc sảo và rõ ràng. Một graphic trong Windows gồm có nhiều đốm nhỏ, mỗi đốm, được gọi là một **pixel**, có khả năng hiển thị 16, 256, ... màu khác nhau.

#### 1. Độ phân giải (resolution)

Thông thường độ phân giải (resolution) của màn hình ta dùng là 800x600, tức là chiều ngang có 800 pixels và chiều cao có 600 pixels. Sau này, để xem các hình rõ hơn ta còn dùng độ phân giải 1028x768 với card màn hình SuperVGA và màn hình tốt. Ta nói card SuperVGA có đến 2MB RAM, tại sao phải cần đến 2MB để hiển thị các hình ảnh đồ họa đẹp?

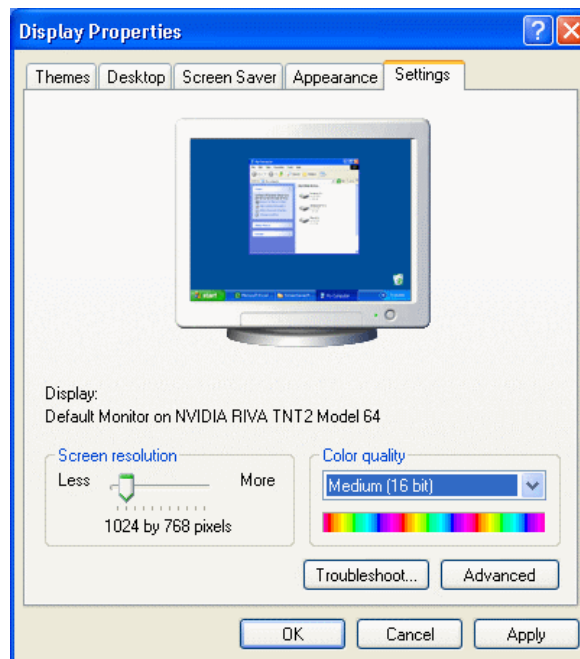
Nếu màu của mỗi pixel được biểu diễn bởi một byte dữ liệu thì với một byte ta có thể chứa một con số từ 0 đến 255, ví dụ: người ta quy ước rằng số 0 tượng trưng cho màu đen, số 255 tượng trưng cho màu trắng chẳng hạn. Nếu độ phân giải của màn hình là 1024x768 thì ta sẽ cần  $1024 \times 768 = 786432$  bytes, tức là gần 0,8 MB.

Một byte có 8 bits. Đôi khi ta nghe nói 16 bit màu, điều này có nghĩa là thay vì 1 byte người ta dùng đến 2 bytes cho mỗi pixel. Như vậy mỗi pixel này có khả năng hiển thị  $2^{16} = 65536$  màu khác nhau. Muốn dùng 16 bit màu cho SuperVGA, ta cần phải có  $1024 \times 768 \times 2 = 1572864$  bytes, tức là gần 1,6 MB. Đó là lý do tại sao ta cần 2MB RAM. Lưu ý là RAM của VGA card (Vector Graphic Adapter) không liên hệ gì với RAM của bộ nhớ máy tính.

Nên nhớ rằng cùng một hình ảnh đồ họa hiển thị trên hai màn hình có cùng độ phân giải, ví dụ như 800x600, nhưng kích thước khác nhau, ví dụ như 14 inches và 17 inches, thì dĩ nhiên hình trên màn hình 17 inches sẽ lớn hơn, nhưng nó vẫn có cùng một số pixels, có điều pixel của nó lớn hơn pixel của màn hình 14 inches.

Nói một cách khác, nếu ta dùng màn hình lớn hơn thì ảnh đồ họa sẽ lớn hơn nhưng không có nghĩa là nó rõ hơn. Muốn thấy rõ chi tiết, ta phải làm cho đồ họa có độ phân giải cao hơn. Ta thay đổi **Display Properties** của một màn hình bằng cách nhấp chuột phải lên desktop rồi

chọn **Properties**, kế đó nhấn vào Tab **Settings** rồi chọn **Screen resolution** và **Color quality** giống như hình dưới đây:

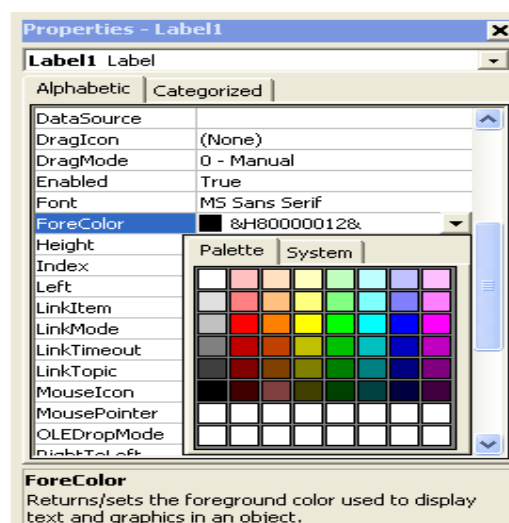


Khi ta tăng độ phân giải của màn hình, các hình ảnh sẽ nhỏ lại vì kích thước của pixel được thu nhỏ lại. Do đó, ta có thể cho hiển thị nhiều thứ hơn trên desktop. Chất lượng của các ảnh đồ họa vẫn không thay đổi, mặc dầu hình nhỏ hơn. Chúng ta cần lưu ý là muốn hình rõ hơn thì khi xây dựng và lưu trữ ảnh đồ họa, ta phải dùng một độ phân giải cao.

## 2. Màu (color)

Khi ta dùng chỉ có một bit (chỉ có trị số 0 hay 1) cho mỗi pixel thì ta chỉ có trắng hay đen. Lúc ấy ta có thể dùng một byte (8 bit) cho 8 pixels. Dầu vậy, nếu độ phân giải của ảnh đủ cao thì hình cũng đẹp. Visual Basic 6 cho ta chỉ định một con số vào mỗi màu VB có thể hiển thị, hay chọn trực tiếp một màu từ Dialog.

Có bốn cách để chỉ định màu, ví dụ: chúng ta chỉ định trực tiếp một con số hay chọn một màu từ bảng màu:



Chúng ta cũng có thể chọn một trong các hằng số định nghĩa sẵn trong VB, gọi là **intrinsic color constants**, chẳng hạn như **vbRed**, **vbBlue**. Danh sách của intrinsic color constants lấy từ VB6 online help được liệt kê dưới đây:

Constant	Value	Description
<b>vbBlack</b>	0x0	Black
<b>vbRed</b>	0xFF	Red
<b>vbGreen</b>	0xFF00	Green
<b>vbYellow</b>	0xFFFF	Yellow
<b>vbBlue</b>	0xFF0000	Blue
<b>vbMagenta</b>	0xFF00FF	Magenta
<b>vbCyan</b>	0xFFFF00	Cyan
<b>vbWhite</b>	0xFFFFFFFF	White

Dùng hàm **QBColor** để chọn một trong 16 màu. Hàm QBColor xuất phát từ thời Quick Basic (QBasic) và trong QBasic chúng ta có thể dùng các con số 1, 2, 3... để chỉ định các màu Blue, Green, Cyan... Hàm QBColor đơn giản hóa cách dùng màu, người sử dụng không cần phải bận tâm về cách trộn ba loại màu căn bản Red, Green, Blue. Chúng ta viết code một cách đơn giản như sau:

```
Label1.ForeColor = QBColor(2)
QBColor(Color As Integer) As Long
```

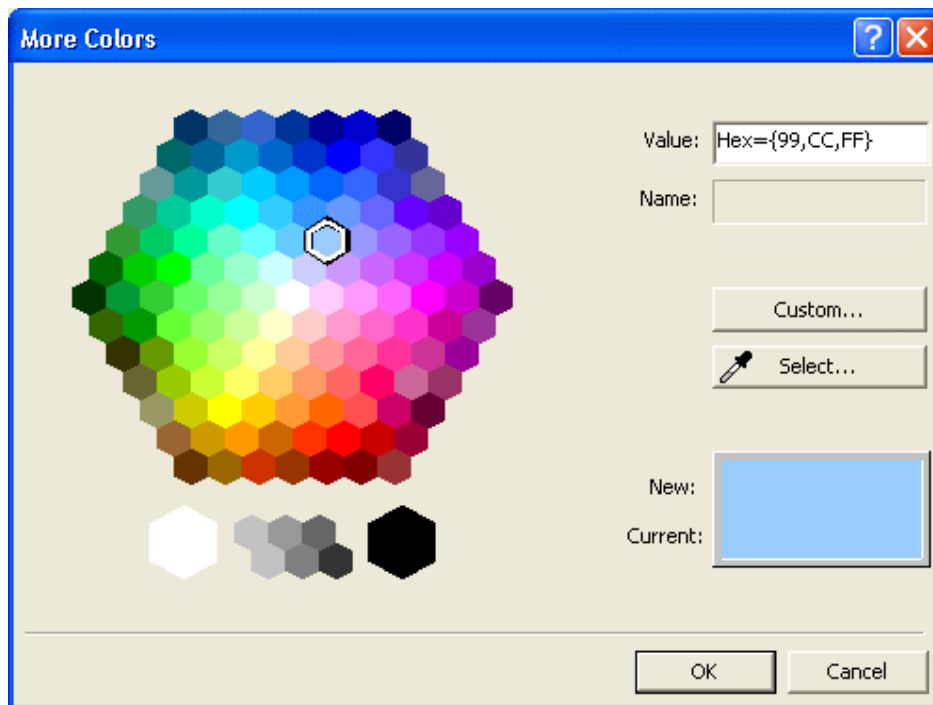
Dưới đây là trị số các màu ta có thể dùng với hàm QBColor.

Trị số	Màu	Trị số	Màu
0	Black	8	Gray
1	Blue	9	Light Blue
2	Green	10	Light Green
3	Cyan	11	Light Cyan
4	Red	12	Light Red
5	Magenta	13	Light Magenta
6	Yellow	14	Light Yellow
7	White	15	Bright White

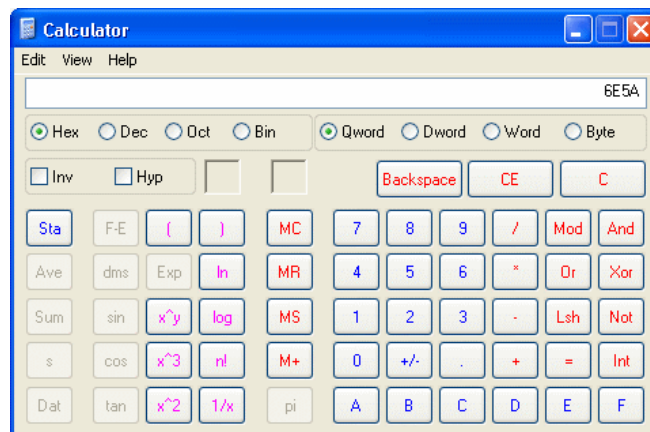
Dùng hàm **RGB** để trộn ba màu **Red**, **Green** và **Blue**. Trong bảng liệt kê các intrinsic color constants phía trên, nếu để ý chúng ta sẽ thấy vbWhite(0xFFFFFFFF) là tổng số của vbRed(0x0000FF), vbGreen(0x00FF00) và vbBlue(0xFF0000). Một màu được biểu diễn bằng sự pha trộn của ba thành phần màu căn bản, mỗi màu bằng một byte có trị số từ 0 đến 255.

Hệ thống số ta dùng hàng ngày là hệ thập phân (decimal). Trị số 0xFF của vbRed là con số 255 viết dưới dạng thập lục phân (Hexadecimal hay còn gọi là **Hex** và ở đây được đánh dấu bằng **0x** trước con số để phân biệt với số thập phân). Trong hệ thống số Hex ta đếm từ 0 đến 9 rồi A,B,C,D,E,F rồi qua số dòng thập lục 10, 11,..., 19, 1A, 1B, ..1E, 1F, 20, 21... Tức là thay vì chỉ dùng 10 ký hiệu từ 0 đến 9 trong hệ thập phân, ta dùng 16 ký hiệu từ 0..9, A..F.

Trong hình dưới đây là một ví dụ cho thấy màu xanh nhạt đã được chọn gồm ba thành phần Blue(0x990000= 153\*256\*256), Green(0xCC00= 204\*256) và Red(0xFF= 255):



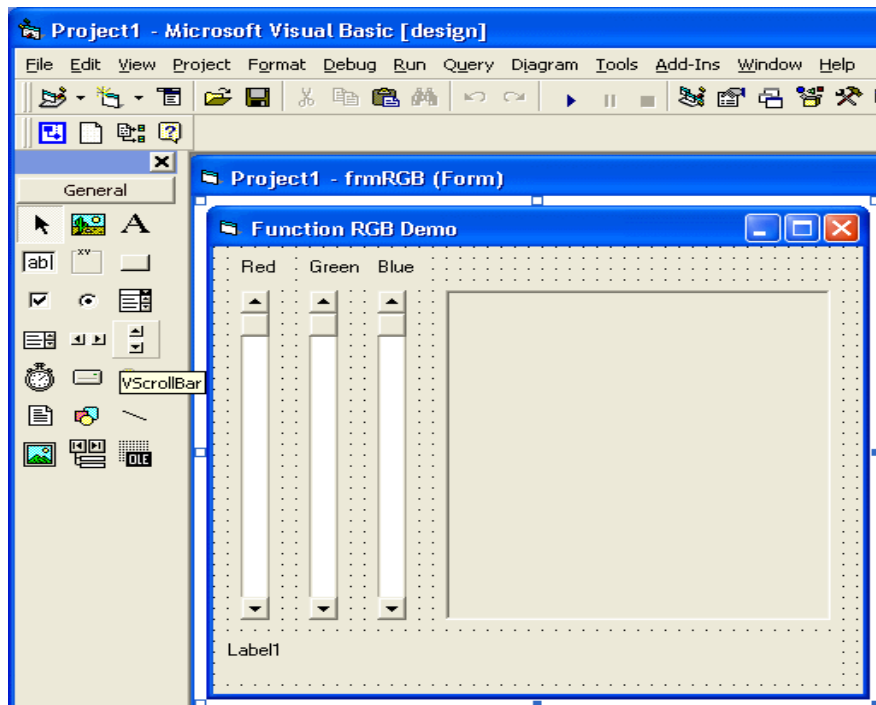
**Ghi chú:** Chúng ta có thể dùng Windows Calculator để hoán chuyển số giữa các dạng Decimal, Binary và Hexadecimal. Chọn **View|Scientific** thay vì **View|Standard**.



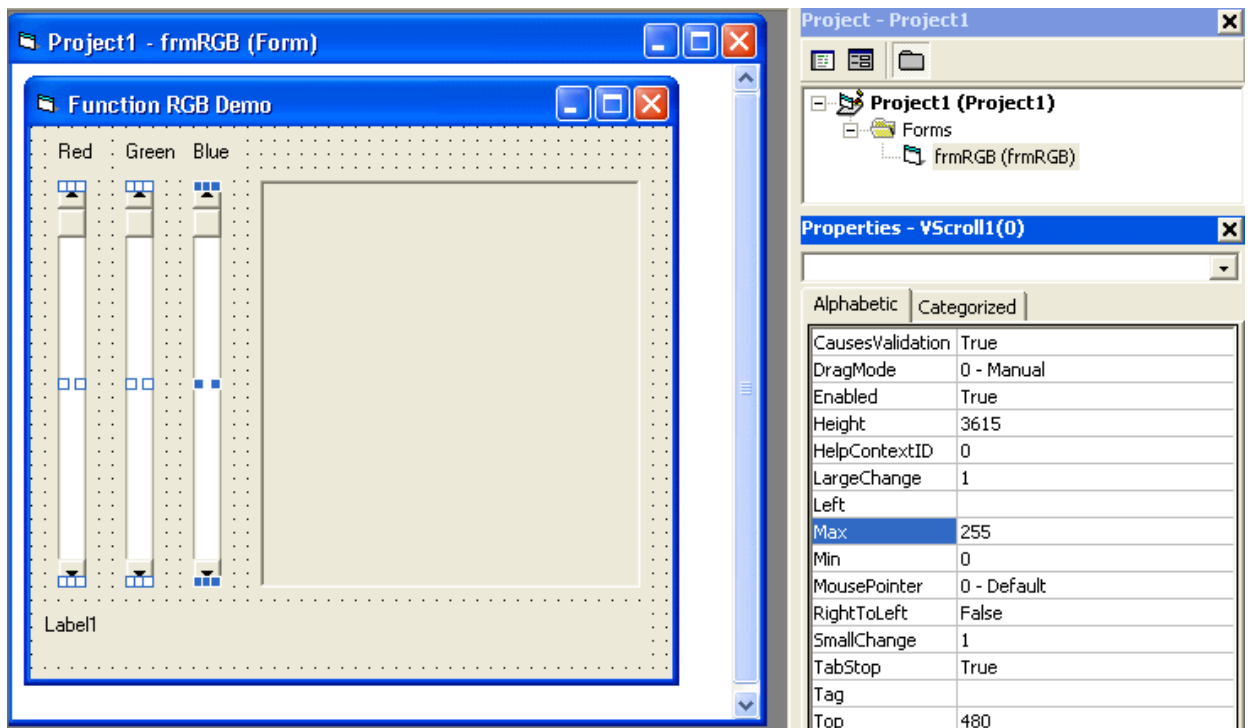
## II. Hàm RGB

Để áp dụng Hàm RGB, ta sẽ viết một chương trình VB6. Chúng ta hãy khởi động một chương trình VB6 mới, bỏ vào một Label tên Label1 với Caption **Red** và một Vertical Scroll tên **VScroll1**. Kế đó select cả hai Label1 và VScroll1 rồi Copy và Paste hai lần để là thêm hai cặp. Đổi Caption của hai Label mới này ra **Green** và **Blue**. Bây giờ ta có một Array ba Vertical Scrolls cùng tên VScroll1, với index là 0,1 và 2.

Đặt một PictureBox tên **picColor** vào bên phải ba VScrolls. Thêm một Label phía dưới, đặt tên nó là **lbIRGBValue**, nhớ clear caption của nó, đừng có để chữ Label1 như dưới đây:



Bây giờ chọn cả ba VScrolls và thay đổi giá trị của **property Max** trong cửa sổ Properties thành **255**, mục đích là để khi kéo thanh điều chỉnh của một VScroll1 lên xuống ta giới hạn trị số của nó từ Min là 0 đến Max là 255.



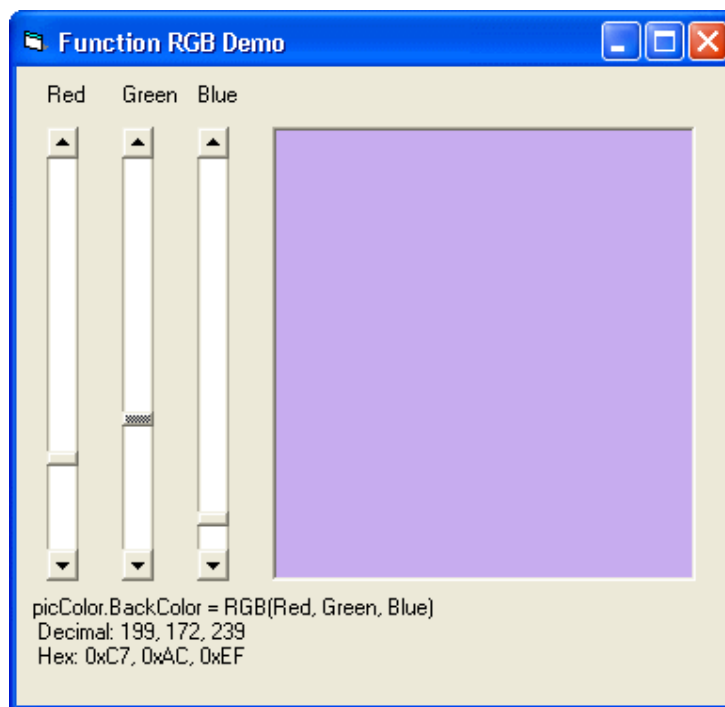
Nội dung chính ta phải làm là viết mã chương trình để xử lý **Event Change** của các VScrolls. Vì chúng là một Array nên ta có thể dùng một Sub duy nhất để handle events đến từ cả ba VScrolls. Mỗi lúc một trong 3 VScrolls thay đổi trị số ta sẽ trộn ba màu Red, Green, Blue biểu diễn bởi trị số của 3 VScrolls thành màu **BackColor** của PictureBox **picColor**. Đồng thời ta cho hiển thị trị số của ba thành phần màu Red, Green và Blue trong Label **lblRGBValue**. Chúng ta hãy double click lên một trong 3 VScrolls rồi viết code như sau:

```

Private Sub VScroll1_Change(Index As Integer)
    ' Use Function RGB to mix 3 colors VScroll1(0) for Red,
    ' VScroll1(1) for Green and VScroll1(2) for Blue
    ' and assign the result to BackColor of PictureBox picColor
    picColor.BackColor = RGB(VScroll1(0).Value, VScroll1(1).Value,
VScroll1(2).Value)
    ' Variable used to prepare hiển thị string
    Dim strRGB As String
    ' Description of what is hiển thị
    strRGB = "picColor.BackColor = RGB(Red, Green, Blue) " &
vbCrLf
    ' Values of Red, Green, Blue in Decimal
    strRGB = strRGB & " Decimal: " & VScroll1(0).Value & ", " &
VScroll1(1).Value & ", " & VScroll1(2).Value & vbCrLf
    ' Values of Red, Green, Blue in Hexadecimal
    strRGB = strRGB & " Hex: 0x" & Hex(VScroll1(0).Value) & ", 0x"
& Hex(VScroll1(1).Value) & ", 0x" & Hex(VScroll1(2).Value)
    ' Assign the resultant string to caption of Label lblRGBValue
    lblRGBValue.Caption = strRGB
End Sub

```

Chúng ta hãy khởi động chương trình rồi nắm các bar của 3 VScrolls kéo lên, kéo xuống để xem kết quả. Cửa sổ của chương trình sẽ có dạng giống như dưới đây:



### III. Một số kỹ thuật

#### 1. Color Mapping

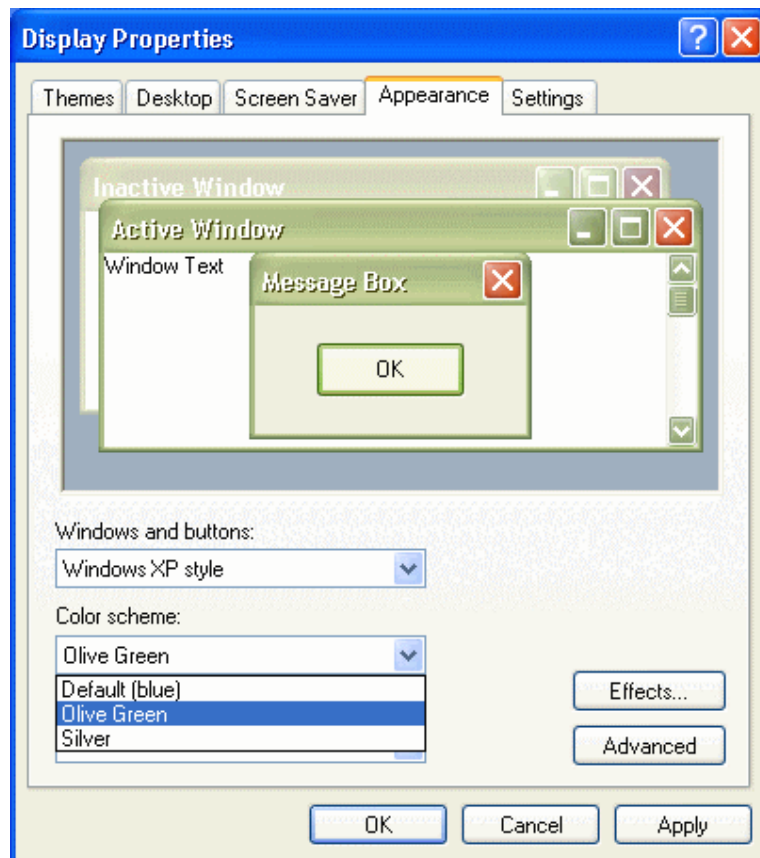
Nếu dùng Hex Calculator đổi con số 0xFFFFFFFF ra decimal ta sẽ được 16777215, nếu kể cả số 0 ta sẽ có tổng cộng 16777216 màu. Lúc này ta bàn về 8bít (1 byte) và 16bít (2 bytes) color, nhưng ở đây ta nói chuyện 3 byte color. Như thế có thể màn hình không đủ khả năng để cung

cấp mọi màu mà hàm RGB tính ra. Vậy với những máy tính sử dụng VGA card thì ta sẽ làm thế nào?

Ví dụ một VGA card chỉ hỗ trợ đến 8 bits và nó chỉ có khả năng cung cấp 256 màu khác nhau. Nếu hàm RGB đòi hỏi một màu mà VGA card có thể cung cấp chính xác thì tốt, nếu không nó sẽ tìm cách dùng hai hay ba đốm màu gần nhau để trộn màu và cho ta ảo tưởng màu ta muốn. Công tác này được gọi là **Color Mapping** và màu được làm ra được gọi là **custom color**.

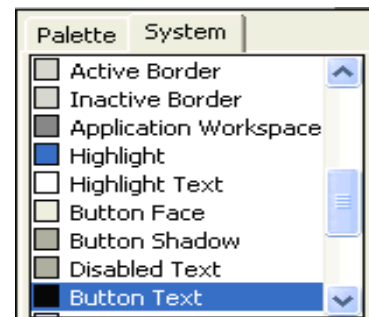
## 2. Dùng Intrinsic Color Constants

Một trong những features của MSWindows là cho ta chọn Color Scheme của Windows theo sở thích. Bình thường, Color Scheme của Windows là Blue, nhưng ta có thể chọn Olive Green hay Silver, nếu ta muốn.



Chỉ tuy nhiên nếu ta đã dùng một màu đỏ đậm để hiển thị tuyệt đẹp thứ gì trong chương trình VB6 mà bây giờ người sử dụng tự nhiên thay đổi Color Scheme thành Olive Green chẳng hạn khiến cho màu đỏ đậm ấy coi chẳng giống ai trong Color Scheme mới.

Để tránh trường hợp ấy, thay vì nói thẳng ra là màu gì (xanh hay đỏ) ta nói dùng màu **vbActiveTitlebar** hay **vbDesktop**, .v.v. Dùng Intrinsic Color Constant sẽ bảo đảm màu ta dùng sẽ được biến đổi theo Color Scheme mà người sử dụng chọn để khỏi bị trường hợp màu trở nên chẳng giống ai. Lúc thiết kế, ta cũng có thể chọn Intrinsic Color Constant từ Tab **System** khi chọn màu.



### 3. Tập tin đồ hoạ

Khi một ảnh đồ hoạ được lưu trữ theo dạng số pixels với màu của chúng như đã nói trên thì ta gọi là một **Bit Map** và tên tập tin của nó trong đĩa có phần mở rộng là **BMP**, ví dụ như **House.bmp**. Lưu trữ kiểu này cần rất nhiều bộ nhớ và bất tiện để gửi đi hay hiển thị trên một trang Web. Do đó người ta dùng những kỹ thuật để giảm thiểu lượng bộ nhớ cần để chứa ảnh đồ hoạ nhưng vẫn giữ được chất lượng của hình ảnh. Có hai dạng tập tin đồ hoạ rất thông dụng trên Web, mang tên với phần mở rộng là **JPG** và **GIF**. Đặc biệt với tập tin GIF ta có thể chứa cả ảnh động (animation), tức là một GIF file có thể chứa nhiều hình (gọi là **Frames**) để chúng lần lượt thay nhau hiển thị, tạo cho người xem có cảm giác như một vật đang di động.



## CHƯƠNG 8. KẾT NỐI ĐẾN CƠ SỞ DỮ LIỆU

Từ phiên bản 5.0, Visual Basic đã cung cấp một control để truy cập cơ sở dữ liệu được gọi là Data. Như ta biết, có một bộ xử lý cơ sở dữ liệu của Microsoft gói kèm theo VB6 - đó là Jet Database Engine. Jet Database Engine là công cụ xử lý dữ liệu của hệ quản trị cơ sở dữ liệu MS Access.

Cho đến VB6, Microsoft cho ta ba kỹ thuật chính:

- DAO (Data Access Objects): DAO là kỹ thuật đặc biệt của Microsoft, chỉ để dùng với Jet Database Engine. Nó rất dễ dùng, hiệu quả và tiện lợi nhưng bị giới hạn trong phạm vi MS Access. Dù vậy, nó rất thịnh hành vì rất dễ sử dụng và mang lại hiệu quả cao.
- ODBC (Open Database Connectivity): ODBC được thiết kế để cho phép người sử dụng kết nối với đủ loại cơ sở dữ liệu mà chỉ dùng một phương thức duy nhất. Điều này giảm bớt gánh nặng cho lập trình viên. Ta chỉ cần học một kỹ thuật lập trình duy nhất mà có thể làm việc với bất cứ loại cơ sở dữ liệu nào. Nhất là sau này nếu cần phải thay đổi loại cơ sở dữ liệu, như nâng cấp từ Access lên SQLServer chẳng hạn, thì sự sửa đổi về mã nguồn chương trình rất ít. Khi dùng ODBC chung với DAO, ta có thể cho cơ sở dữ liệu của Access nối với các cơ sở dữ liệu khác. Có một bất lợi của ODBC là hơi phức tạp khi sử dụng.
- RDO (Remote Data Object): Một trong những lý do chính để RDO được thiết kế là giải quyết khó khăn về sự rắc rối của ODBC. Cách lập trình với RDO đơn giản như DAO, nhưng thật ra nó dùng ODBC nên cho phép người sử dụng nối với nhiều cơ sở dữ liệu. Tuy nhiên, RDO không được thịnh hành lắm.

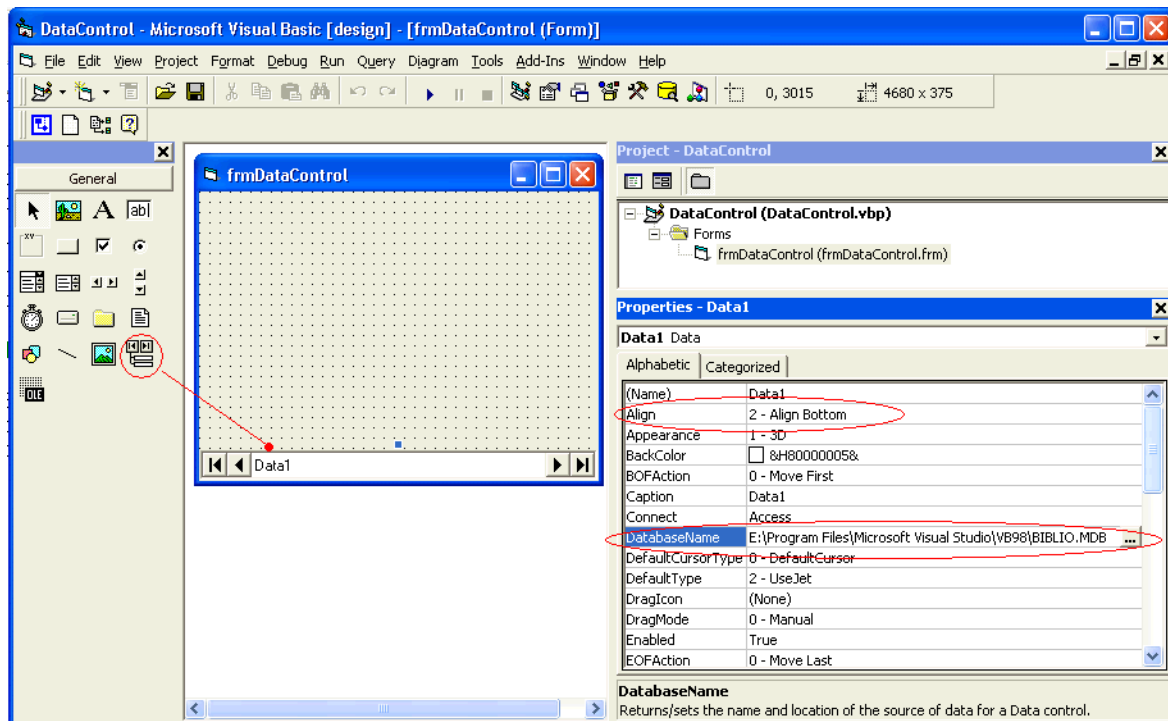
VB6 tiếp tục hỗ trợ các kỹ thuật nói trên, và cho thêm một kỹ thuật truy cập cơ sở dữ liệu mới, rất quan trọng, đó là ADO (ActiveX Data Objects).

### I. Sử dụng Data Control

Cách dùng giản tiện của Data Control là đặt nó lên một Form rồi làm việc với những Properties của nó. Chúng ta hãy bắt đầu một dự án VB6 mới, cho nó tên DataControl bằng cách click tên của dự án trong Project Explorer bên phải rồi thay đổi các thuộc tính trong Properties Window.

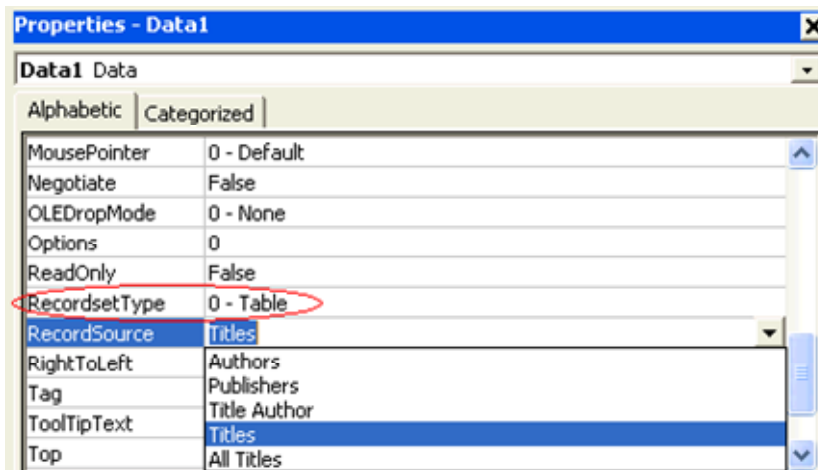
DoubleClick lên Icon của Control Data trong Toolbox. Một Control Data tên Data1 sẽ hiện ra trên Form. Muốn cho nó nằm bên dưới Form, giống như một StatusBar, hãy thiết lập property Align của nó trong Properties Window thành 2 - Align Bottom.

Click bên phải dòng thuộc tính DatabaseName, kế đó click lên nút browse có ba chấm để chọn một file Access database từ giao thoại cho Data1. Ở đây ta chọn E:\Program Files\Microsoft Visual Studio\VB98\BIBLIO.MDB, trong máy tính của chúng ta có thể nó nằm trên disk C hay D.



Trong chương trình này ta muốn làm việc với bảng Titles của cơ sở dữ liệu BIBLIO.MDB, để xem và hiệu chỉnh các bản ghi. Để ý thuộc tính DefaultType của Data1 có trị số 2- UseJet, tức là dùng kỹ thuật DAO, thay vì dùng kỹ thuật ODBC.

Khi chúng ta click lên thuộc tính Recordsource của Data1, rồi click lên cái tam giác nhỏ bên phải, một ComboBox sẽ mở ra cho ta thấy danh sách các bảng trong cơ sở dữ liệu. Chúng ta hãy chọn Titles. Để ý thuộc tính RecordsetType của Data1 có trị số là 0 - Table:



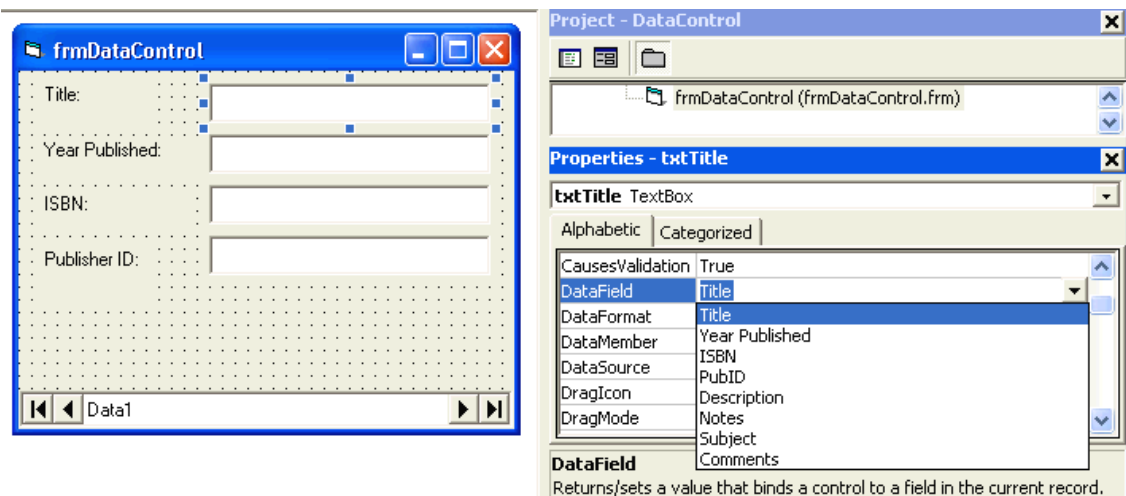
Thuật ngữ mới mà ta sẽ dùng thường xuyên khi truy cập dữ liệu trong VB6 là Recordset (tập hợp các bản ghi). Recordset là một tập hợp các bản ghi, nó có thể chứa một số các bản ghi hay không có bản ghi nào cả. Một bản ghi trong Recordset có thể là một bản ghi lấy từ một Bảng. Trong trường hợp ấy có thể ta lấy về tất cả bản ghi trong bảng hay chỉ những bản ghi thỏa mãn một điều kiện, ví dụ như ta chỉ muốn lấy các bản ghi của những sách xuất bản trước năm 1990 (Year Published < 1990).

Một bản ghi trong Recordset cũng có thể là tập hợp các cột (columns) từ hai hay nhiều bảng qua các mối liên hệ one-to-one và one-to-many. Ví dụ, khi lấy các bản ghi từ bảng Titles, ta muốn có thêm chi tiết tên công ty (Company Name) và điện thoại (Telephone) của nhà xuất bản (bảng Publishers) bằng cách dùng Foreign Key PubID trong bảng Titles làm Primary Key trong bảng Publishers để lấy các chi tiết ấy. Trong trường hợp này ta có thể xem như có một bảng ảo là tập hợp của hai bảng Titles và Publishers.

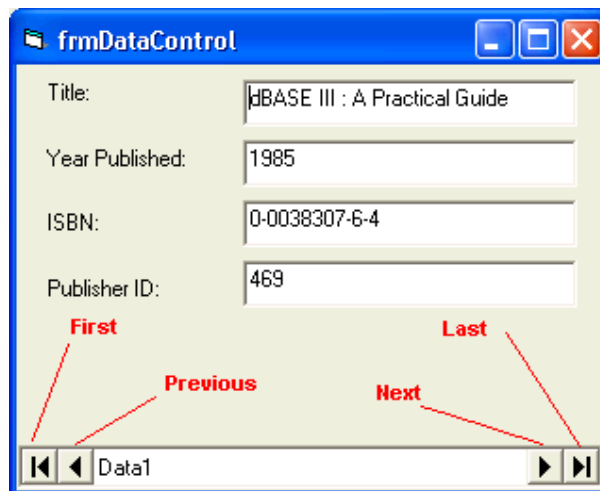
Bây giờ chúng ta hãy đặt lên Form 4 labels với captions: Title, Year Published, ISBN và Publisher ID. Kế đó cho thêm 4 textboxes tương ứng và đặt tên chúng là txtTitle, txtYearPublished, txtISBN và txtPublisherID.

Chọn textbox txtTitle, rồi thiết lập thuộc tính Datasource của nó trong Properties Window thành Data1. Khi click lên thuộc tính Datafield của txtTitle và mở ComboBox ra chúng ta sẽ thấy liệt kê tên các trường trong bảng Titles. Đó là vì Data1 được coi như trung gian lấy từ bảng Titles của cơ sở dữ liệu. Ở đây ta sẽ chọn cột Title.

Lập lại công tác này cho 3 textboxes kia, và chọn các cột Year Published (năm xuất bản), ISBN (số lý lịch trong thư viện quốc tế), và PubID (số lý lịch nhà xuất bản) làm Datafield cho chúng.



Tới đây, mặc dầu chưa viết một dòng code, ta có thể chạy chương trình được rồi. Nó sẽ hiển thị chi tiết của bản ghi đầu tiên trong bảng Titles như dưới đây:



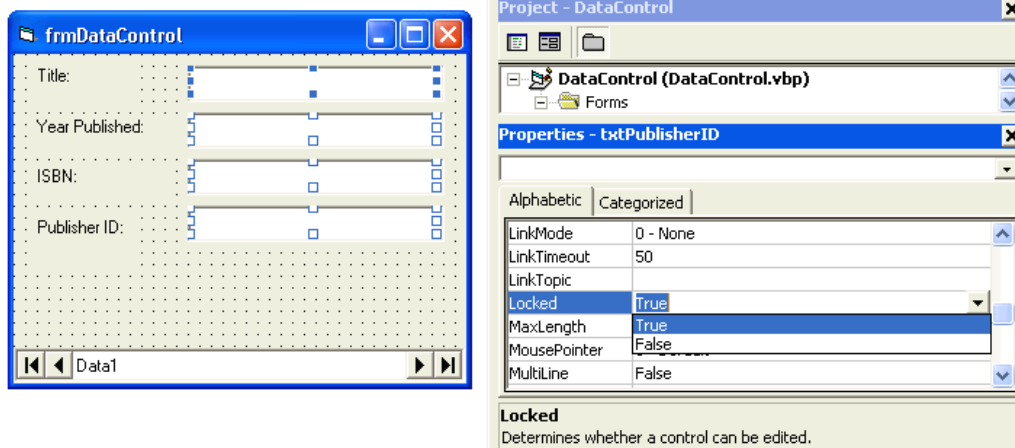
Chúng ta có thể bấm các nút di chuyển Navigator Buttons để đi đến các bản ghi đầu (first), trước (previous), kế (next) và cuối (last). Mỗi lần chúng ta di chuyển đến một bản ghi mới là chi tiết của bản ghi ấy sẽ hiển thị. Nếu không dùng các Navigator Buttons, ta cũng có thể code để làm công tác tương đương bằng cách gọi các Recordset methods MoveFirst, MovePrevious, MoveNext và MoveLast.

Khi bản ghi cuối của Recordset đang hiển thị, nếu ta gọi phương thức MoveLast thì thuộc tính EOF (End-Of-File) của Recordset trở thành True. Tương tự như vậy, khi bản ghi thứ nhất của Recordset đang hiển thị, nếu ta gọi phương thức MovePrevious thì thuộc tính BOF (Begin-Of-File) của Recordset trở thành True. Nếu một Recordset không có chứa một bản ghi nào cả thì cả hai thuộc tính EOF và BOF đều là True.

Đặc tính hiển thị dữ liệu trong các textbox theo đúng bản ghi hiện thời (current record) được gọi là data binding hay data bound (ràng buộc dữ liệu) và control TextBox hỗ trợ chức năng này được nói là Data Aware (nhận biết dữ liệu).

Khi bản ghi đầu tiên đang hiển thị, nếu chúng ta sửa đổi Year Published để đổi từ 1985 thành 1983 rồi nhấn nút lệnh Next để hiển thị bản ghi thứ nhì, kế đó nhấn nút Previous để hiển thị lại bản ghi đầu tiên thì chúng ta sẽ thấy là trường Year Published của bản ghi đầu tiên đã thật sự được thay đổi (updated) thành 1983.

Điều này có nghĩa rằng khi duyệt Data1 từ bản ghi này đến bản ghi khác thì nếu bản ghi này đã có sự thay đổi vì người sử dụng đã sửa đổi, nó lưu trữ sự thay đổi đó trước khi di chuyển. Chưa chắc là chúng ta muốn điều này, do đó, nếu chúng ta không muốn người sử dụng tình cờ sửa đổi một bản ghi thì chúng ta có thể thiết lập thuộc tính Locked của các textboxes ấy thành True để người sử dụng không thể sửa đổi các textboxes như trong hình dưới đây:



## 1. Chỉ định vị trí cơ sở dữ liệu lúc chạy chương trình

Cách chỉ định tên DatabaseName trong giai đoạn thiết kế (at design time) ta đã dùng trước đây tuy tiện lợi nhưng hơi nguy hiểm, vì khi ta cài chương trình này lên máy tính của khách, chưa chắc tập tin cơ sở dữ liệu ấy nằm trong một thư mục có cùng tên. Ví dụ trên máy tính người lập trình thì cơ sở dữ liệu nằm trong thư mục E:\Program Files\Microsoft Visual Studio\VB98, nhưng trên máy tính của khách thì cơ sở dữ liệu nằm trong thư mục C:\VB6\DataControl chẳng hạn. Do đó, khi chương trình khởi động ta nên xác định lại vị trí của cơ sở dữ liệu. Giả dụ ta muốn để cơ sở dữ liệu trong cùng một thư mục với chương trình đang chạy, ta có thể dùng thuộc tính Path của Application Object App như sau:

```
Dim AppFolder As String
```

```

Private Sub Form_Load()
    ' Fetch Folder where this program EXE resides
    AppFolder = App.Path
    ' make sure it ends with a back slash
    If Right(AppFolder, 1) <> "\" Then AppFolder = AppFolder & "\"
    ' Assign Full path database filename to Data1
    Data1.DatabaseName = AppFolder & "BIBLIO.MDB"
End Sub

```

Với cách code nói trên ta sẽ đảm bảo chương trình tìm thấy tập tin cơ sở dữ liệu đúng chỗ mà không cần biết người ta cài chương trình chúng ta ở đâu trong đĩa cứng của máy tính khách.

## 2. Thêm bớt các bản ghi

Chương trình trên dùng cũng tạm được, nhưng nó không cho ta phương tiện để thêm (add), bớt (delete) các bản ghi. Bây giờ chúng ta hãy để vào Form 5 buttons tên: cmdEdit, cmdNew, cmdDelete, cmdUpdate và cmdCancel.

Mặc dầu chúng ta không thấy, nhưng thật ra Control Data Data1 có một thuộc tính Recordset và khi ta dùng Navigator buttons là di chuyển từ bản ghi này đến bản ghi khác trong Recordset ấy. Ta có thể nói đến nó bằng Notation (cách viết) Data1.Recordset, và mỗi lần muốn lấy Recordset mới nhất từ cơ sở dữ liệu ta dùng phương thức Refresh như Data1.Recordset.Refresh.

Lúc chương trình mới khởi động, người sử dụng đang xem (browsing) các bản ghi thì hai buttons Update và Cancel không cần phải làm việc. Do đó ta sẽ nhân tiện Lock (khóa) các textboxes và disable (làm cho bất lực) hai buttons này vì không cần dùng chúng.

Trong Sub SetControls dưới đây, ta dùng một parameter gọi là Editing với trị số False hay True tùy theo người sử dụng đang Browse hay Edit, ta gọi là Browse mode và Edit mode. Trong Edit mode, các Textboxes được unlocked (mở khóa) và các nút cmdNew, cmdDelete và cmdEdit trở nên bất lực:

```

Sub SetControls(ByVal Editing As Boolean)
    ' Lock/Unlock textboxes
    txtTitle.Locked = Not Editing
    txtYearPublished.Locked = Not Editing
    txtISBN.Locked = Not Editing
    txtPublisherID.Locked = Not Editing
    ' Enable/Disable buttons
    CmdUpdate.Enabled = Editing
    CmdCancel.Enabled = Editing
    CmdDelete.Enabled = Not Editing
    cmdNew.Enabled = Not Editing
    CmdEdit.Enabled = Not Editing
End Sub

```

Trong Browse mode, Form có dạng như sau:

Sub SetControls được gọi trong Sub Form\_Load khi chương trình khởi động và trong Sub CmdEdit khi người sử dụng click nút Edit như sau:

```
Private Sub Form_Load()
    ' Fetch Folder where this program EXE resides
    AppFolder = App.Path
    ' make sure it ends with a back slash
    If Right(AppFolder, 1) <> "\" Then AppFolder = AppFolder & "\"
    ' Assign Full path database filename to Data1
    Data1.DatabaseName = AppFolder & "BIBLIO.MDB"
    ' Place controls in Browse Mode
    SetControls (False)
End Sub
Private Sub CmdEdit_Click()
    ' Place controls in Edit Mode
    SetControls (True)
End Sub
```

Khi ta xóa một bản ghi trong recordset, vị trí của bản ghi hiện tại (current record) vẫn không thay đổi. Do đó, sau khi xóa một bản ghi ta phải MoveNext. Tuy nhiên, nếu ta vừa xóa bản ghi cuối của Recordset thì sau khi MoveNext, thuộc tính EOF của Recordset sẽ thành True. Thành ra ta phải kiểm tra điều đó, nếu đúng vậy thì lại phải MoveLast để hiển thị bản ghi cuối của Recordset như trong code của Sub cmdDelete\_Click dưới đây:

```
Private Sub CmdDelete_Click()
    On Error GoTo DeleteErr
    With Data1.Recordset
        ' Delete new record
        .Delete
        ' Move to next record
        .MoveNext
        If .EOF Then .MoveLast
    End With
    Exit Sub
DeleteErr:
    MsgBox Err.Description
    Exit Sub
End Sub
```

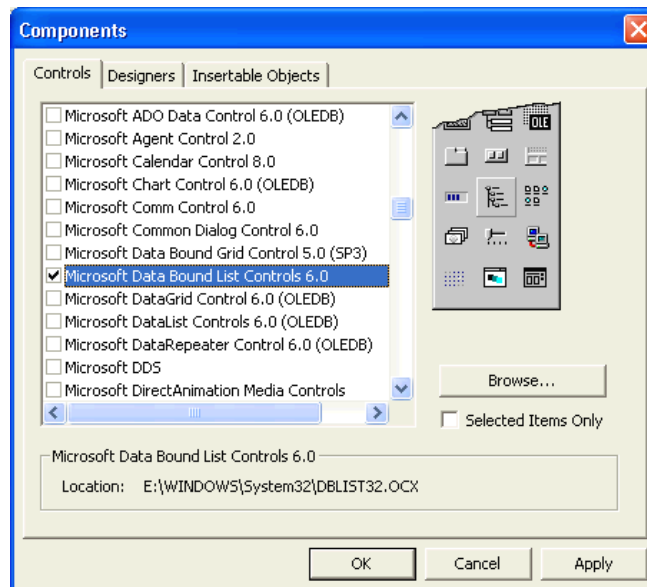
Trong lúc code, ta Update (cập nhật) một bản ghi trong Recordset bằng phương thức Update. Nhưng ta chỉ có thể gọi phương thức Update của một Recordset khi Recordset đang ở trong Edit hay AddNew mode. Ta đặt một Recordset vào Edit mode bằng cách gọi phương thức Edit của Recordset, ví dụ như Data1.Recordset.Edit. Tương tự như vậy, ta đặt một Recordset vào AddNew mode bằng cách gọi phương thức AddNew của Recordset, ví dụ như Data1.Recordset.AddNew.

```
Private Sub cmdNew_Click()  
    ' Place Recordset into Recordset AddNew mode  
    Data1.Recordset.AddNew  
    ' Place controls in Edit Mode  
    SetControls (True)  
End Sub
```

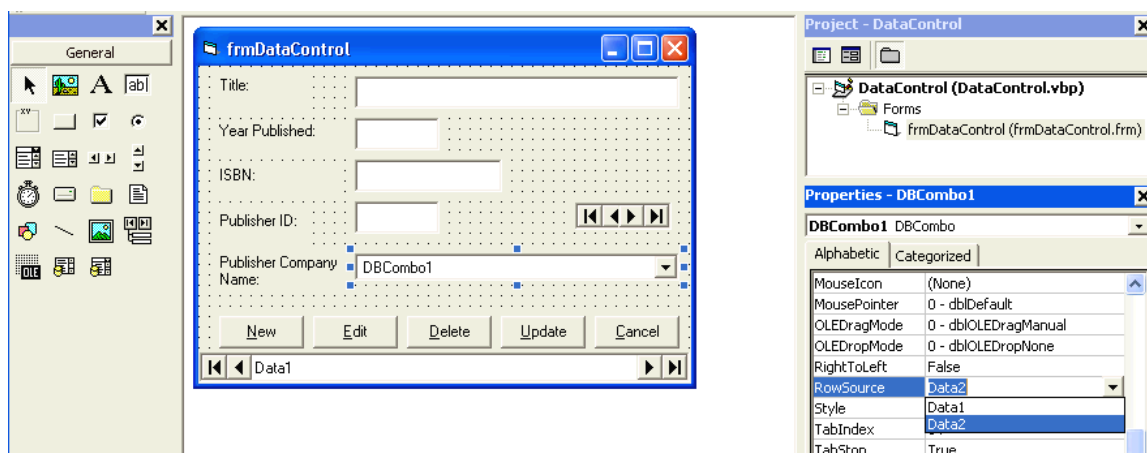
Sau khi Recordset gọi phương thức Update thì Recordset ấy ra khỏi AddNew hay Edit modes. Ta cũng có thể tự thoát ra khỏi AddNew hay Edit modes, hay nói cho đúng hơn là hủy bỏ mọi pending (đang chờ đợi) Update bằng cách gọi phương thức CancelUpdate, ví dụ như Data1.Recordset.CancelUpdate.

### 3. Dùng DataBound Combo

Trong chương trình hiện tại ta chỉ hiển thị lý lịch nhà xuất bản (PubID) của Title, chớ không có thêm chi tiết. Nếu chương trình lưu trữ PubID, nhưng hiển thị được Company Name của nhà xuất bản cho ta làm việc để khỏi phải nhớ các con số thì sẽ tốt hơn. Ta có thể thực hiện điều đó bằng cách dùng Control DBCombo (Data Bound Combo). Chúng ta hãy dùng IDE Menu Command Project | Components... để chọn Microsoft Data Bound List Controls 6.0 rồi click Apply.



Kế đó, thêm một DBCombo tên DBCombo1 vào Form. Vì ta cần một Recordset khác để cung cấp Bảng Publisher cho DBCombo1, nên chúng ta hãy thêm một control Data thứ nhì tên Data2 vào Form. Cho Data2, hãy thiết lập thuộc tính DatabaseName thành E:\Program Files\Microsoft Visual Studio\VB98\BIBLIO.MDB và thuộc tính RecordSource thành Publishers. Để không cho người ta thấy hình Data2 lúc run-time, chúng ta hãy thiết lập Visible nó thành False.



Mục đích của chúng ta khi dùng DBCombo1 là hiển thị Company Name của nhà xuất bản, nhưng đằng sau lưng thì không có gì thay đổi, tức là ta vẫn làm việc với PubID cho các record Title của Data1. Khi người sử dụng click lên DBCombo1 để chọn một nhà xuất bản, thì ta theo Company Name đó mà chứa PubID tương ứng trong record Title của Data1. Do đó có nhiều thứ ta phải sắp đặt cho DBCombo1 như sau:

Thuộc tính	Giá trị	Chú thích
RowSource	Data2	Đây là datasource của chính DBCombo1. Nó cung cấp bảng Publishers.
Listfield	Company Name	Khi RowSource phía trên đã được chọn rồi, Combo của thuộc tính Listfield này sẽ hiển thị các trường của bảng Publishers. Company Name là trường của RowSource mà ta muốn hiển thị trên DBCombo1.
DataSource	Data1	Đây là datasource của bản ghi mà ta muốn. sửa đổi, tức là bản ghi của bảng Titles
Datafield	PubID	Trường (của record Title) sẽ được thay đổi.
BoundColumn	PubID	Trường trong RowSource (bảng Publishers) tương ứng với item user chọn trong DBCombo1 (Company Name).

Khi trong Edit mode user chọn một Company Name khác trong DBCombo1 rồi click nút Update chúng ta sẽ thấy Textbox txtPublisherID cũng đổi theo và hiển thị con số lý lịch PubID mới. Nếu trước khi Update chúng ta muốn thấy PubID mới hiển thị trong Textbox txtPublisherID thì chúng ta có thể dùng Event Click của DBCombo1 như sau:

```
Private Sub DBCombo1_Click(Area As Integer)
    ' Hiển thị new PuBID
    txtPublisherID.Text = DBCombo1.BoundText
End Sub
```

Thuộc tính BoundText của DBCombo1 là trị số của BoundColumn mà ta có thể truy cập (viết hay đọc) được. Ví dụ như chúng ta muốn mỗi khi thêm một bản ghi Title mới thì default PubID là 324, tức là Company Name= "GLOBAL ENGINEERING". Chúng ta có thể assign trị số 324 vào thuộc tính BoundText của DBCombo1 trong Sub cmdNew\_Click như sau:

```
Private Sub cmdNew_Click()
    ' Place Recordset into Recordset AddNew mode
```



```

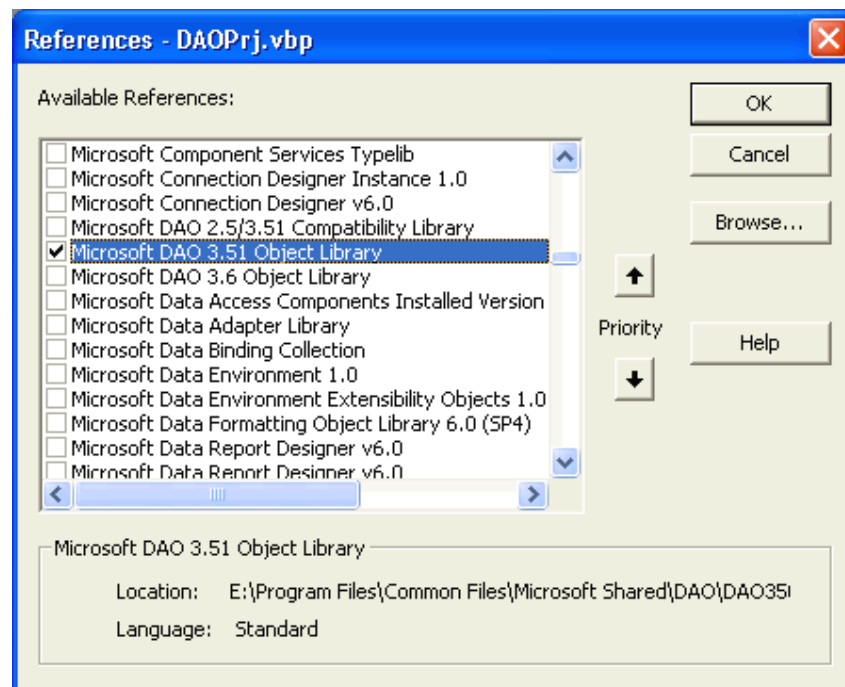
Data1.Recordset.AddNew
' Default Publisher is "GLOBAL ENGINEERING", i.e. PubID=324
DBCombo1.BoundText = 324
' Place controls in Edit Mode
SetControls (True)
End Sub

```

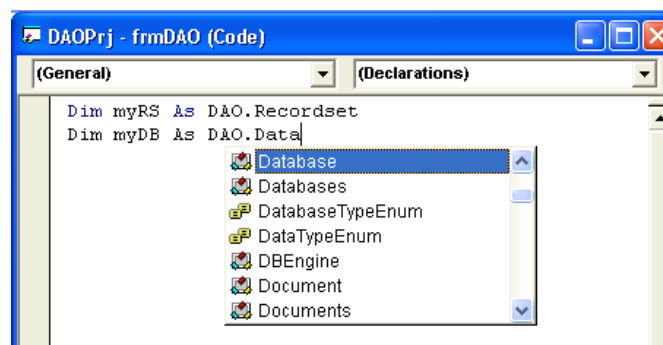
## II. Lập trình với kỹ thuật DAO

### 1. Tham chiếu DAO

Trong phần này chúng tôi giới thiệu những bước lập trình căn bản với MS Access Database qua kỹ thuật DAO mà không cần dùng đến Control *Data* như trong phần trước. Ta sẽ cần đến vài Objects trong thư viện DAO, do đó nếu chúng ta mở một dự án VB6 mới thì hãy dùng Menu Command Project | References... để chọn Microsoft DAO 3.51 Object Library bằng cách click lên checkbox bên trái như trong hình dưới đây.



Sau đó trong code của Form chính ta sẽ khai báo biến myDatabase cho một instance của DAO database và biến myRS cho một DAO recordset. Ở đây ta nói rõ Database và Recordset là thuộc loại DAO để phân biệt với Database và Recordset thuộc loại ADO (ActiveX Data Object) sau này.



Bây giờ chúng ta hãy đặt lên Form chính, tên frmDAO, 4 labels với captions: Title, Year Published, ISBN và Publisher ID. Kế đó cho thêm 4 textboxes tương ứng và đặt tên chúng là txtTitle, txtYearPublished, txtISBN và txtPublisherID.

Điều ta muốn làm là khi Form mới được loaded, nó sẽ lấy về từ cơ sở dữ liệu một Recordset chứa tất cả bản ghi trong bảng Titles theo thứ tự về mẫu tự (alphabetical order) của field Title và hiển thị bản ghi đầu tiên.

## 2. Dùng keyword SET

Công việc đầu tiên cần làm là mở một Database Object dựa vào tên đầy đủ (full path name) của Access database:

```
' Open main database
Set myDB = OpenDatabase(AppFolder & "BIBLIO.MDB")
```

Đề ý chữ Set trong câu lệnh được viết ở trên. Đó là vì myDB là một Pointer đến một Object. Mặc dầu từ đây về sau ta sẽ dùng myDB như một Database theo cách giống như bất cứ biến thuộc kiểu dữ liệu nào khác, nhưng khi chỉ định lần đầu là nó từ đâu đến thì ta dùng chữ Set, để nói rằng thật ra myDB không phải là Object Database, nhưng là Pointer đến Object Database.

Mục đích là VB6 dành ra cho khi cần thực hiện một phần trong bộ nhớ để chứa Object Database khi ta nhận được nó từ sự thực thi của Phương thức OpenDatabase. Mặc dù vị trí chỗ chứa Object Database trong bộ nhớ không nhất định, nhưng vì ta nắm con trỏ chỉ đến vị trí ấy nên ta vẫn có thể làm việc với nó một cách bình thường. Con trỏ đó là value (trị số) của biến myDB. Vì giá trị này không phải là Object, nhưng nó chứa memory address chỉ đến (point to hay refer to) Object Database, nên ta gọi nó là Pointer.

Lập trình dùng Pointer nói chung rất linh động và mang lại hiệu quả cao trong các ngôn ngữ như C, Pascal, C++ ,v.v.. Tuy nhiên, lập trình viên phải nhớ trả lại hệ điều hành phần bộ nhớ mình dùng khi không còn cần nó nữa để hệ điều hành lại lấy chỗ cho Object khác. Nếu công việc quản lý dùng lại bộ nhớ không ổn thỏa thì có những mảnh bộ nhớ nằm rải rác mà hệ điều hành không biết. Dần dần hệ điều hành sẽ không còn bộ nhớ để cấp phát cho các hoạt động tiếp theo nữa. Ta gọi hiện tượng ấy là memory leakage (hao mòn bộ nhớ). Các ngôn ngữ sau này như Java, C# đều không dùng Pointer nữa. Visual Basic không muốn lập trình viên dùng Pointer. Chỉ trong vài trường hợp đặc biệt VB6 mới lộ ra cho ta thấy thật ra ở trong hậu trường VB6 Runtime dùng Pointer như trong trường hợp này.

Tương tự như vậy, vì Recordset là một Pointer đến một Object, ta cũng dùng Set khi chỉ định một DAO Recordset lấy về từ Phương thức OpenRecordset của cơ sở dữ liệu myDB.

```
'Open recordset
Set myRS=myDB.OpenRecordset("Select*from Titles ORDER BY Title")
```

Cái parameter loại String ta dùng cho phương thức OpenRecordset là một Lệnh (Statement) SQL. Nó chỉ định cho cơ sở dữ liệu lấy tất cả mọi trường (columns) (**Select \***) của mỗi bản ghi từ Bảng Titles (**from Titles**) làm một Recordset và sắp xếp các bản ghi trong Recordset ấy theo alphabetical order của trường Title (**ORDER BY Title**).

Nhớ là Recordset này cũng giống như thuộc tính Recordset của một Control Data mà ta dùng trong phần trước. Bây giờ có Recordset rồi, ta có thể hiển thị chi tiết của bản ghi đầu tiên nếu Recordset ấy có ít nhất một bản ghi. Ta kiểm tra điều ấy dựa vào thuộc tính RecordCount của Recordset như trong code dưới đây:

```
Private Sub Form_Load()
```

```

' Fetch Folder where this program EXE resides
AppFolder = App.Path
' make sure it ends with a back slash
If Right(AppFolder, 1) <> "\" Then AppFolder = AppFolder & "\"
' Open main database
Set myDB = OpenDatabase(AppFolder & "BIBLIO.MDB")
'Open recordset
Set myRS=myDB.OpenRecordset("Select * from Titles ORDER BY
Title")
' if Recordset is not empty then hiển thị the first record
If myRS.RecordCount > 0 Then
    myRS.MoveFirst ' move to first record
    Hiển thịrecord ' hiển thị details of current record
End If
End Sub

```

Sau khi dùng phương thức MoveFirst của Recordset để định vị con trỏ hiện tại ở bản ghi đầu tiên, ta hiển thị trị số các trường của bản ghi bằng cách ghi giá trị chúng vào các textboxes của Form như sau:

```

Private Sub Hiển thịrecord()
' Assign record fields to the appropriate textboxes
With myRS
    ' Assign field Title to textbox txtTitle
    txtTitle.Text = .Fields("Title")
    txtYearPublished.Text = .Fields("[Year Published]")
    txtISBN.Text = .Fields("ISBN")
    txtPublisherID.Text = .Fields("PubID")
End With
End Sub

```

Để ý vì trường Year Published gồm có hai chữ nên ta phải đặt tên của trường ấy giữa hai dấu ngoặc vuông ([]). Để tránh bị phiền phức như trong trường hợp này, khi chúng ta đặt tên trường trong lúc thiết kế một bảng hãy dán dính các chữ lại với nhau, đừng để rời ra. Ví dụ, nên dùng YearPublished thay vì Year Published.

### 3. Các nút di chuyển

Muốn có các nút duyệt tương đương với của một Control Data, chúng ta hãy đặt lên Form 4 buttons mang tên CmdFirst, CmdPrevious, CmNext và CmdLast với captions: <<, <, >, >>.

Viết mã lệnh cho các nút này cũng đơn giản, nhưng ta phải coi chừng khi người sử dụng muốn di chuyển quá bản ghi cuối cùng hay bản ghi đầu tiên. Ta phải kiểm tra xem EOF có trở thành True khi người sử dụng click CmdNext, hay BOF có trở thành True khi người sử dụng click CmdPrevious:

```

Private Sub CmdNext_Click()
    myRS.MoveNext ' Move to next record
    ' Display record details if has not gone past the last record
    If Not myRS.EOF Then
        Displayrecord ' hiển thị details of current record
    Else
        myRS.MoveLast ' Move back to last record
    End If
End Sub

```

```

Private Sub CmdPrevious_Click()
    myRS.MovePrevious ' Move to previous record
    ' Display record details if has not gone past the first record
    If Not myRS.BOF Then
        Displayrecord ' hiển thị details of current record
    Else
        myRS.MoveFirst ' Move back to first record
    End If
End Sub

Private Sub CmdFirst_Click()
    myRS.MoveFirst ' Move back to first record
    Displayrecord ' hiển thị details of current record
End Sub

Private Sub CmdLast_Click()
    myRS.MoveLast ' Move back to last record
    Displayrecord ' hiển thị details of current record
End Sub

```

Khi chạy chương trình chúng ta sẽ thấy nó hiển thị chi tiết của bản ghi đầu tiên khác với trong phần trước đây vì các bản ghi đã được sắp xếp:

#### 4. Thêm bớt các bản ghi

Giống như chương trình trong phần trước, ta sẽ thêm các nút chọn để thêm (add), bớt (delete) các bản ghi. Bây giờ chúng ta hãy để vào Form 5 các nút lệnh có tên: cmdEdit, cmdNew, cmdDelete, cmdUpdate và cmdCancel.

Chỗ nào trong chương trình trước ta dùng Data1.Recordset thì bây giờ ta dùng myRS.

Ta sẽ dùng lại Sub SetControls với parameter Editing có trị số False hay True tùy theo người sử dụng đang Browse hay Edit. Trong Browse mode, các Textboxes bị Locked (khóa) và các nút cmdUpdate và cmdCancel trở nên bất lực. Trong Edit mode, các Textboxes được unlocked (mở khóa) và các nút cmdNew, cmdDelete và cmdEdit trở nên bất lực.

Vì ở đây không có Data Binding nên đợi cho đến khi Update ta mới đặt Recordset vào AddNew hay Edit mode. Do đó ta chỉ cần nhớ là khi người sử dụng edits là đang sửa đổi một bản ghi hiện hữu hay thêm một bản ghi mới. Ta chứa trị số Boolean ấy trong variable AddNewRecord. Nếu người sử dụng sắp thêm một bản ghi mới thì AddNewRecord = True, nếu người sử dụng sắp sửa đổi một bản ghi hiện hữu thì AddNewRecord = False.

Ngoài ra, khi người sử dụng sắp thêm một bản ghi mới bằng cách click nút New thì ta phải tự clear (làm trắng) hết các textboxes bằng cách xác nhận Empty vào thuộc tính Text của chúng như sau:

```
' If Editing existing record then AddNewRecord = False
' Else AddNewRecord = true
Dim AddNewRecord As Boolean

Private Sub ClearAllFields()
    ' Clear all the textboxes
    txtTitle.Text = ""
    txtYearPublished.Text = ""
    txtISBN.Text = ""
    txtPublisherID.Text = ""
End Sub

Private Sub cmdNew_Click()
    ' Remember that this is Adding a new record
    AddNewRecord = True
    ' Clear all textboxes
    ClearAllFields
    ' Place controls in Edit Mode
    SetControls (True)
End Sub
Private Sub CmdEdit_Click()
    ' Place controls in Edit Mode
    SetControls (True)
    ' Remember that this is Editing an existing record
    AddNewRecord = False
End Sub
```

Nếu người sử dụng nhấn Cancel trong khi đang edit các textboxes, ta không cần gọi phương thức CancelUpdate vì Recordset chưa bị đặt vào AddNew hay Edit mode. Ở đây ta chỉ cần hiển thị lại chi tiết của current record, tức là hủy bỏ những gì người sử dụng đang đánh vào:

```
Private Sub CmdCancel_Click()
    ' Cancel update
    SetControls (False)
    ' Redisplay details or current record
    Displayrecord
End Sub
```

Lúc người sử dụng nhấn Update, chúng ta có dịp để kiểm tra data xem có trường nào bị bỏ trống (nhất là Primary Key ISBN bắt buộc phải có trị số) hay có gì không valid bằng cách gọi Function GoodData. Nếu GoodData trả lại một trị số False thì ta không xúc tiến với việc Update. Nếu GoodData trả về trị số True thì ta đặt Recordset vào AddNew hay Edit mode tùy theo trị số của Boolean variable AddNewRecord.

Giống như khi hiển thị chi tiết của một bản ghi ta phải ghi từng trường vào textbox, thì bây giờ khi Update ta phải làm ngược lại, tức là ghi giá trị Text của từng textbox vào Record Field tương ứng. Sau cùng ta gọi phương thức Update của recordset và cho các controls trở lại Browse mode:

```
Private Function GoodData() As Boolean
    ' Check Data here. If Invalid Data then GoodData = False
```

```

        GoodData = True
    End Function
    Private Sub CmdUpdate_Click()
        ' Verify all data, if Bad then do not Update
        If Not GoodData Then Exit Sub
        ' Assign record fields to the appropriate textboxes
        With myRS
            If AddNewRecord Then
                .AddNew ' Place Recordset in AddNew Mode
            Else
                .Edit ' Place Recordset in Edit Mode
            End If
            ' Assign text of txtTitle to field Title
            .Fields("Title") = txtTitle.Text
            .Fields("[Year Published]") = txtYearPublished.Text
            .Fields("ISBN") = txtISBN.Text
            .Fields("PubID") = txtPublisherID.Text
            ' Update data
            .Update
        End With
        ' Return controls to Browse Mode
        SetControls (False)
    End Sub

```

Cũng vì không có Data Binding, nên khi người sử dụng xóa một bản ghi, sau khi di chuyển qua bản ghi kế tiếp ta phải tự hiển thị chi tiết của bản ghi đó như sau:

```

    Private Sub CmdDelete_Click()
        On Error GoTo DeleteErr
        With myRS
            .Delete ' Delete new record
            .MoveNext ' Move to next record
            If .EOF Then .MoveLast
            Displayrecord ' Display details of current record
        End With
        Exit Sub
    DeleteErr:
        MsgBox Err.Description
        Exit Sub
    End Sub

```

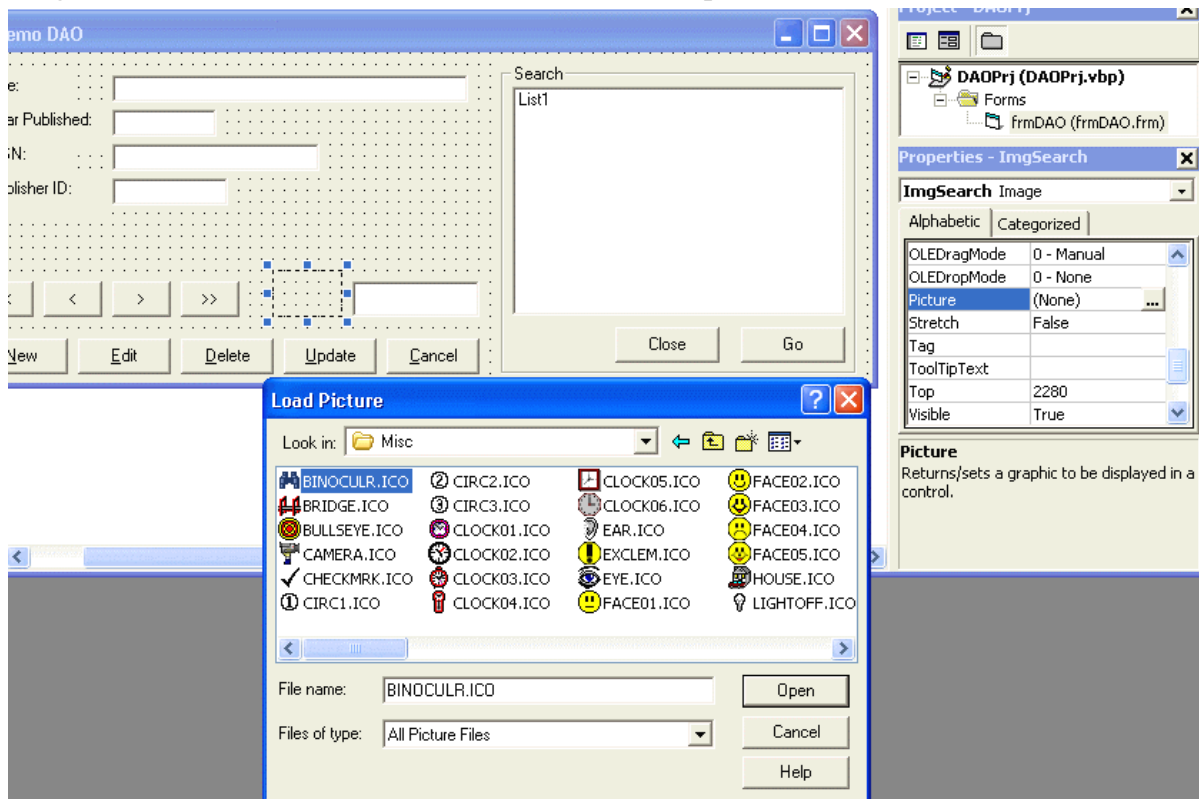
## 5. Tìm một bản ghi

Tiếp theo đây, ta muốn liệt kê các sách có tiêu đề chứa một chữ hay câu nào đó, ví dụ như chữ "Guide". Kể đó người sử dụng có thể chọn một sách bằng cách chọn tiêu đề sách ấy và click nút Go. Chương trình sẽ định vị đến bản ghi của sách ấy và hiển thị chi tiết của nó.

Bây giờ chúng ta hãy cho vào Form một textbox tên txtSearch và một Image tên ImgSearch. Kế đó đặt một frame tên fraSearch vào Form. Để lên frame này một listbox tên List1 để hiển thị tiêu đề các sách, và hai buttons tên CmdClose và CmdGo, với nhãn là Close và Go. Sau khi lựa chọn một cuốn sách trong List1, người sử dụng sẽ click nút Go để hiển thị chi tiết sách ấy. Nếu đổi ý, người sử dụng sẽ click nút Close để làm biến mất frame fraSearch.

Bình thường frame fraSearch chỉ hiện ra khi cần, nên lúc đầu hãy thiết lập Visible của nó thành False. Ta sẽ cho ImgSearch hiển thị hình một ống dòm nên chúng ta hãy click vào bên

phải thuộc tính Picture trong Properties Window để chọn Icon BINOCULR.ICO từ folder E:\Program Files\Microsoft Visual Studio\Common\Graphics\Icons\Misc:



Primary Key của bảng Titles là ISBN. Khi người sử dụng lựa chọn một cuốn sách, ta cần phải biết ISBN của sách ấy để định vị nó trong Recordset myRS. Do đó, trong khi thêm tiêu đề của một sách vào List1, ta đồng thời thêm ISBN của sách ấy vào một Listbox thứ hai tên List2. Ta chỉ sẽ dùng List2 sau hậu trường, nên hãy thiết lập thuộc tính Visible của nó thành False. Dưới đây là code để load tiêu đề sách và ISBN vào các Listboxes:

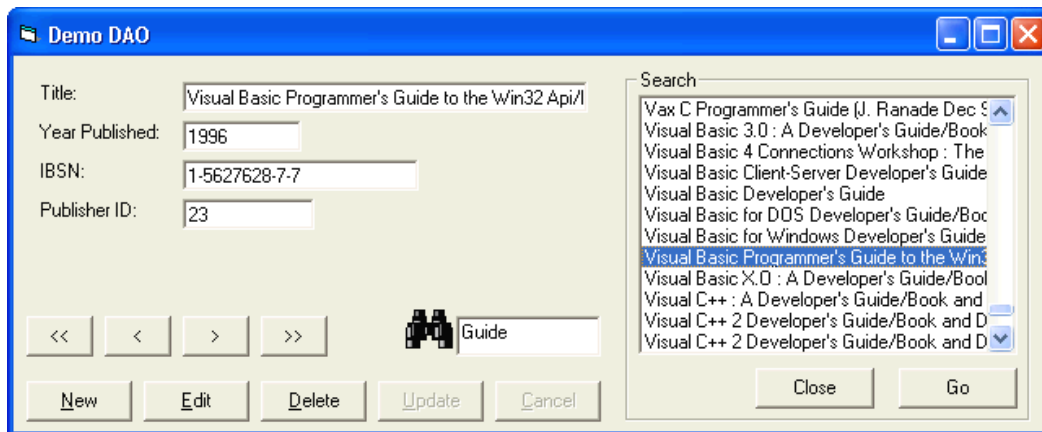
```
Private Sub ImgSearch_Click()
    ' Show Search Frame
    fraSearch.Visible = True
    Dim SrchRS As DAO.Recordset
    Dim SQLCommand As String
    ' Define SQL statement
    SQLCommand = "Select * from Titles where Title LIKE '" & "*" & txtSearch & "*" & "' ORDER BY Title"
    ' Fetch all records having Title containing the text pattern
    given by txtSearch
    Set SrchRS = myDB.OpenRecordset(SQLCommand)
    ' If Recordset is not Empty then list the books' titles in
    List1
    If SrchRS.RecordCount > 0 Then
        List1.Clear ' Clear List1
        ' We use List2 to contain the Primary Key ISBN
        corresponding to the books in List1
        List2.Clear ' Clear List2
        With SrchRS
            ' Iterate through the Recordset until EOF
            Do While Not SrchRS.EOF
```

```

        ' Hiển thị Title in List1
        List1.AddItem .Fields("Title")
        ' Store corresponding ISBN in List2
        List2.AddItem .Fields("ISBN")
        .MoveNext ' Move to next record in the Recordset
    Loop
End With
End If
End Sub

```

Khi người sử dụng Click ImgSearch với text pattern là chữ Guide, ta sẽ thấy hình dưới đây:



Trong SELECT statement bên trên ta dùng toán tử LIKE trên text pattern, chữ Guide, có wildcard character (\*) ở hai bên. Wildcard character là chỗ có (hay không có) chữ gì cũng được. Trong trường hợp này có nghĩa là chỉ cần có chữ Guide trong tiêu đề sách là được, không cần biết nó nằm ở đâu. Ngoài ra sự chọn lựa này không có Case Sensitive, tức là chữ guide, Guide hay GUIDE đều được cả.

Khi người sử dụng nhấn nút Go, ta sẽ dùng phương thức FindFirst của Recordset myRS để định chỗ của bản ghi có trị số Primary Key là dòng text trong List2 tương ứng với tiêu đề được chọn trong List1 như sau:

```

Private Sub CmdGo_Click()
    Dim SelectedISBN As String
    Dim SelectedIndex As Integer
    Dim Criteria As String
    ' Index of line selected by user in List1
    SelectedIndex = List1.ListIndex
    ' Obtain corresponding ISBN in List2
    SelectedISBN = List2.List(SelectedIndex)
    ' Define Search criteria - use single quotes for selected text
    Criteria = "ISBN = '" & SelectedISBN & "'"
    ' Locate the record, it will become the current record
    myRS.FindFirst Criteria
    ' Hiển thị details of current record
    Hiển thịrecord
    ' Make fraSearch disappeared
    fraSearch.Visible = False
End Sub

```

Lưu ý là trong Criteria ta dùng xâu ký tự, vì ISBN thuộc loại text, chứ không phải là một con số, nên ta phải đặt nó giữa hai dấu ngoặc đơn.



## 6. Bookmark

Khi di chuyển từ bản ghi này đến bản ghi khác trong Recordset, đôi khi ta muốn đánh dấu vị trí của một bản ghi để có dịp sẽ trở lại. Ta có thể thực hiện điều ấy bằng cách ghi nhớ Bookmark của Recordset.

Ví dụ: khi người sử dụng nhấn nút Go, ta muốn nhớ vị trí của bản ghi lúc ấy để sau này quay trở lại khi người sử dụng nhấn nút Go Back. Chúng ta hãy thêm vào Form một nút lệnh tên CmdGoBack với Caption Go Back. Ta sẽ thêm một variable tên LastBookmark loại data type Variant:

```
Dim LastBookMark As Variant
```

Lúc đầu nút lệnh CmdGoBack invisible, và chỉ trở nên visible sau khi người sử dụng nhấn nút Go. Ta thêm các dòng codes sau vào Sub CmdGo\_Click() như sau:

```
' Remember location of current record
LastBookMark = myRS.BookMark
CmdGoback.Visible = True
```

Dưới đây là code để quay trở lại vị trí current record trước đây trong Recordset:

```
Private Sub CmdGoback_Click()
    ' Reposition record to last position
    myRS.BookMark = LastBookMark
    ' Rehiển thị details or current record
    Displayrecord
End Sub
```

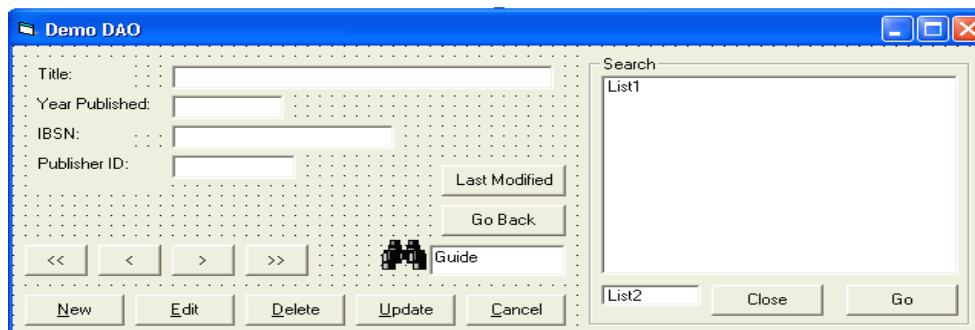
## 7. LastModified

LastModified là vị trí của bản ghi vừa mới được sửa đổi hay thêm vào trong Recordset. Để thử điều này chúng ta hãy thêm một nút lệnh không hiển thị có tên CmdLastModified với caption là Last Modified. Nút lệnh này chỉ hiện ra sau khi người sử dụng nhấn Update.

Bất cứ lúc nào chúng ta nhấn nút CmdLastModified, bản ghi mới vừa được sửa đổi hay thêm vào sẽ hiển thị:

```
Private Sub CmdLastModified_Click()
    ' Reposition record to last position
    myRS.BookMark = myRS.LastModified
    ' Redisplay details or current record
    Displayrecord
End Sub
```

Dưới đây là hình của Form lúc đang được thiết kế:



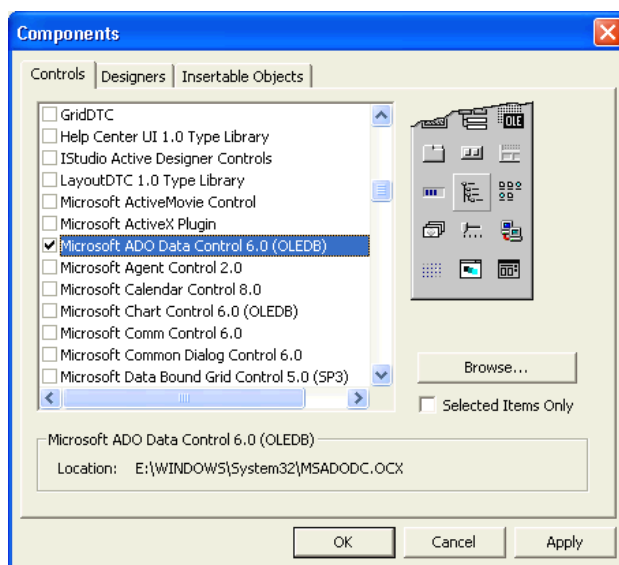
### III. Lập trình với ADO

Visual Basic 6 cho ta nhiều sự lựa chọn về kỹ thuật khi lập trình với cơ sở dữ liệu, hoặc là dùng DAO như trong hai phần trước, hoặc là dùng ADO (ActiveX Data Objects).

Sự khác biệt chính giữa ADO và DAO là ADO cho phép ta làm việc với mọi loại nguồn dữ liệu, không nhất thiết phải là cơ sở dữ liệu của Access hay ODBC. Nguồn dữ liệu có thể là danh sách các địa chỉ Email, hay một tập tin văn bản, trong đó mỗi dòng là một bản ghi gồm những trường ngăn cách bởi các dấu phẩy.

Nếu trong DAO ta dùng thẳng tên của MSAccess Database thì trong ADO cho ta nối với một cơ sở dữ liệu qua một Connection bằng cách chỉ định một Connection String. Trong Connection String có Database Provider (ví dụ như Jet, ISAM, Oracle, SQLServer..v.v.), tên Cơ sở dữ liệu, UserName/Password để đăng nhập vào một cơ sở dữ liệu... Sau đó ta có thể lấy về những recordsets, và cập nhật hóa các bản ghi bằng cách dùng những lệnh SQL trên các bảng hay dùng những thủ tục lưu trữ bên trong cơ sở dữ liệu.

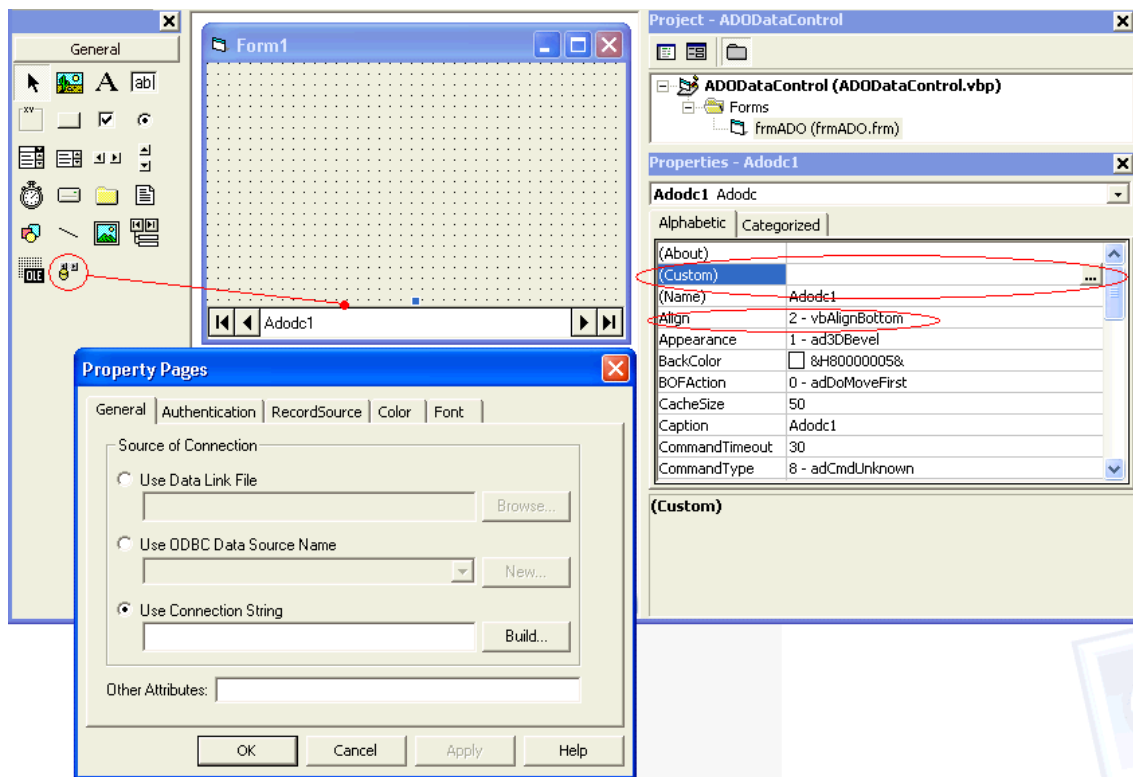
Bình thường, khi ta mới khởi động một dự án VB6 mới, Control *Data ADO* không có sẵn trong IDE. Muốn có nó, chúng ta hãy dùng Menu Command Project | Components..., rồi chọn Microsoft ADO Data Control 6.0 (OLEDB) từ giao diện Components như dưới đây:



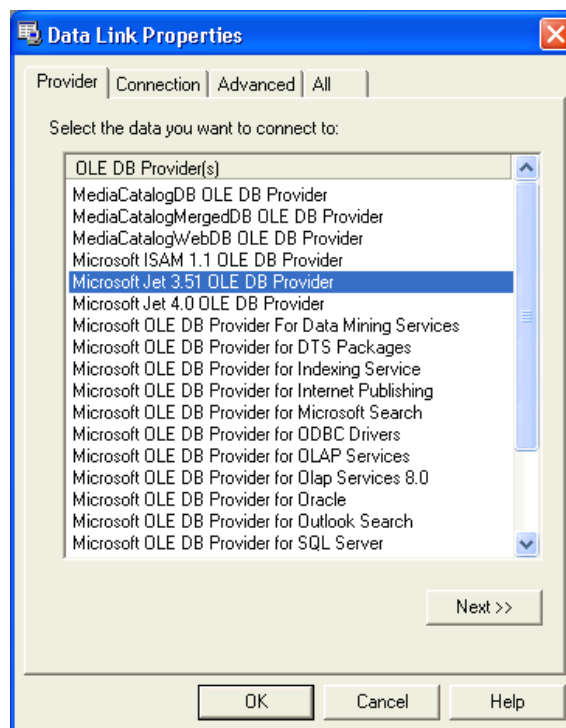
Chúng ta hãy bắt đầu một dự án VB6 mới, cho nó tên ADODataControl bằng cách click tên project trong Project Explorer bên phải rồi thay đổi thuộc tính Name trong Properties Window. Sửa tên của form chính thành frmADO, và đánh câu ADO DataControl Demo vào Caption của nó.

DoubleClick lên Icon của Control Data ADO trong Toolbox. Một Control Data ADO tên Adodc1 sẽ hiện ra trên Form. Muốn cho nó nằm bên dưới Form, giống như một StatusBar, hãy set property Align của nó trong Properties Window thành 2 - vbAlignBottom.

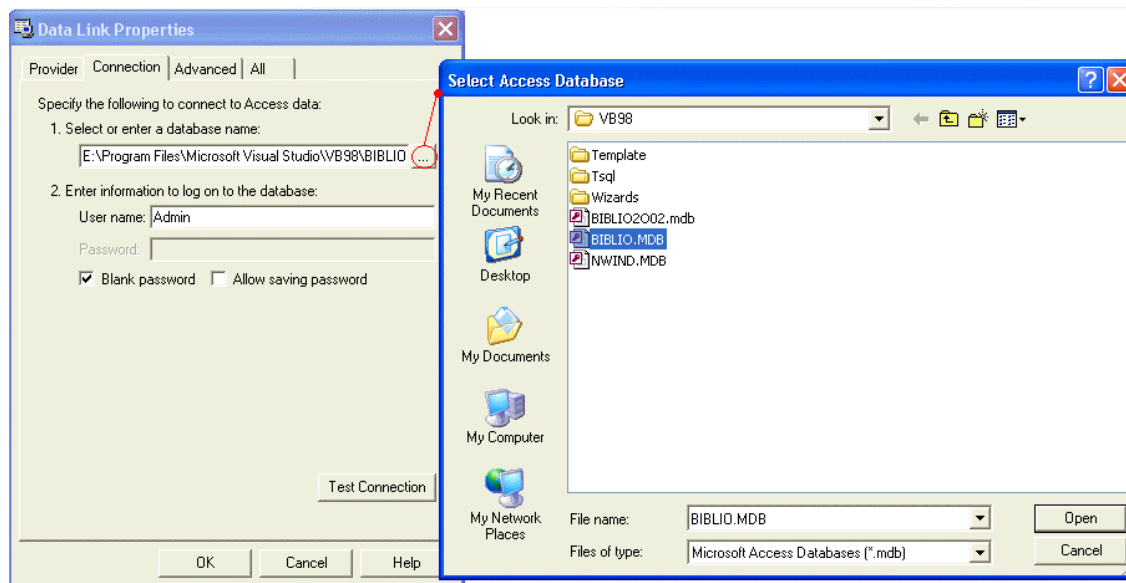
Click bên phải dòng property (Custom), kế đó click lên nút browse có ba chấm để giao thoại Property Pages hiện ra. Trong giao thoại này, trên Tab General chọn Radio (Option) Button Use Connection String rồi click nút Build....



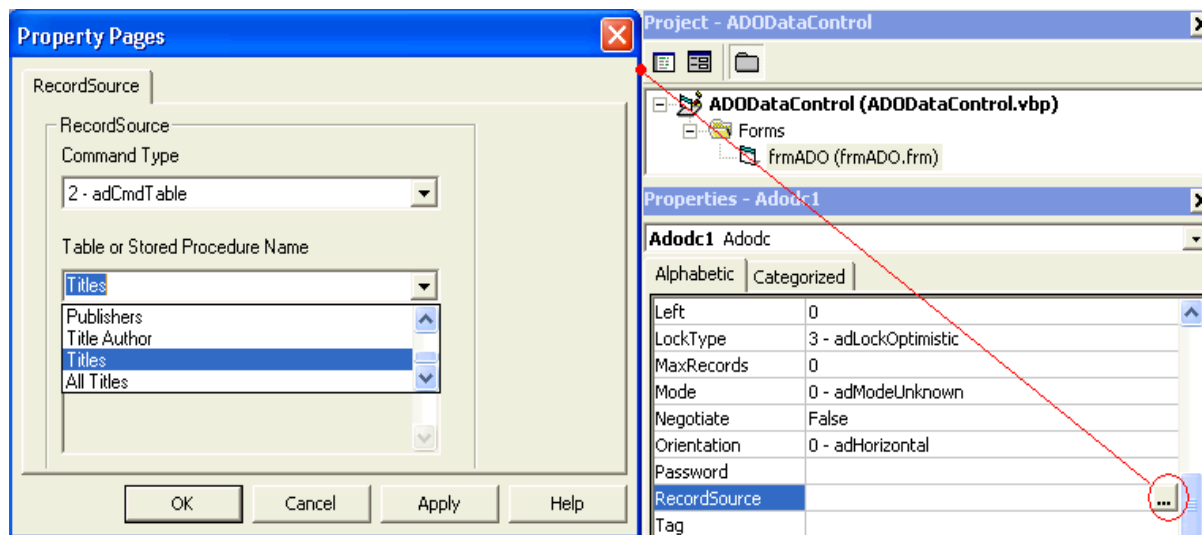
Trong giao thoại Data Link Properties, Tab Provider, chọn Microsoft Jet 3.51 OLE DB Provider, rồi click nút Next >> hay Tab Connection.



Ở chỗ Select or enter a database name ta chọn E:\Program Files\Microsoft Visual Studio\VB98\BIBLIO.MDB, trong máy tính của chúng ta có thể file ấy nằm trên disk C hay D. Sau đó, chúng ta có thể click nút Test Connection phía dưới để thử xem connection có được thiết lập tốt không.



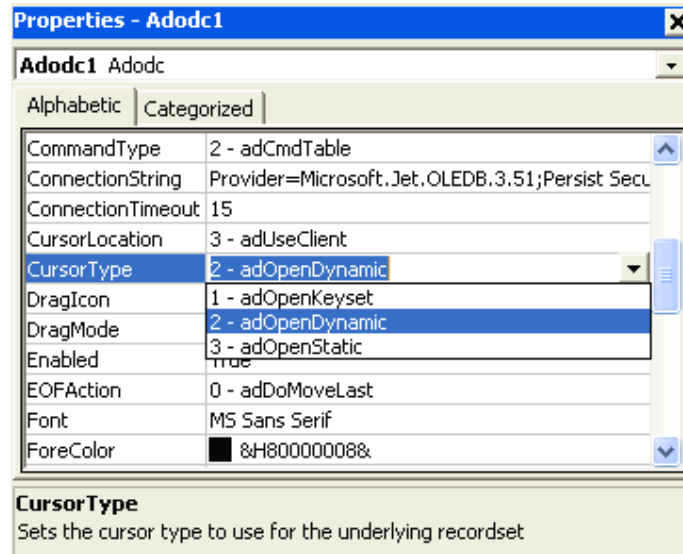
Lập connection xong rồi, ta chỉ định muốn lấy gì về làm Recordset bằng cách chọn thuộc tính Recordsource của Adodc1. Trong giao diện Property Pages của nó chọn 2-adCmdTable làm Command Type, kế đó mở Combo box cho Table or Stored Procedure Name để chọn bảng Titles.



Tùy theo cách ta dùng Recordset trong ADO, nó có ba loại và được gọi là Cursor Type. Cursor chẳng qua là một tên khác của Recordset:

- Static Cursor: Static Cursor cho chúng ta một static copy (bản sao cứng ngắt) của các bản ghi. Trong lúc chúng ta dùng Static Cursor, nếu có ai khác sửa đổi hay thêm, bớt gì vào recordset chúng ta sẽ không thấy.
- Keyset Cursor: Keyset Cursor hơn Static Cursor ở chỗ trong lúc chúng ta dùng nó, nếu có ai sửa đổi bản ghi nào chúng ta sẽ biết. Nếu ai xóa bản ghi nào, chúng ta sẽ không thấy nó nữa. Tuy nhiên chúng ta sẽ không biết nếu có ai thêm một bản ghi nào vào recordset.
- Dynamic Cursor: Như chữ sống động (dynamic) hàm ý, trong lúc chúng ta đang dùng một Dynamic Cursor, nếu có ai khác sửa đổi hay thêm, bớt gì vào recordset chúng ta sẽ thấy hết.

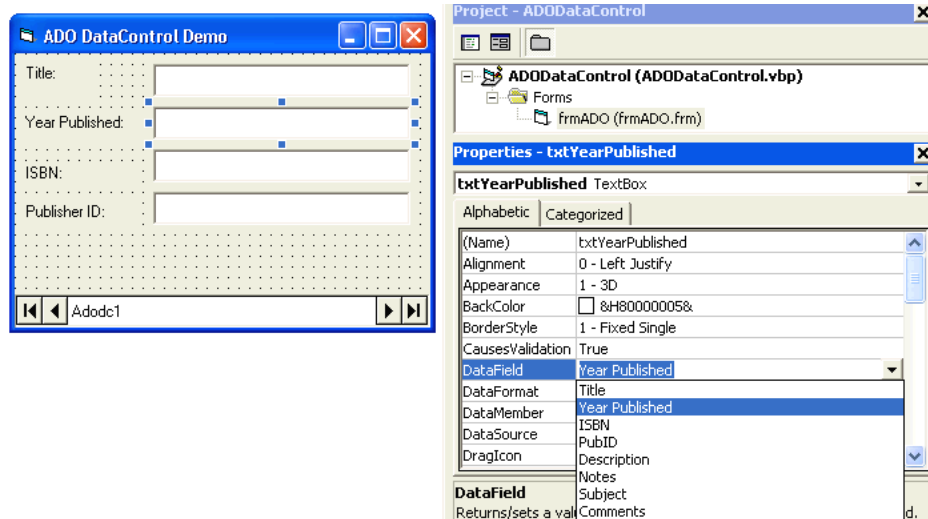
Chúng ta hãy chọn trị số 2-adOpenDynamic cho property Cursor Type của Adodc1:



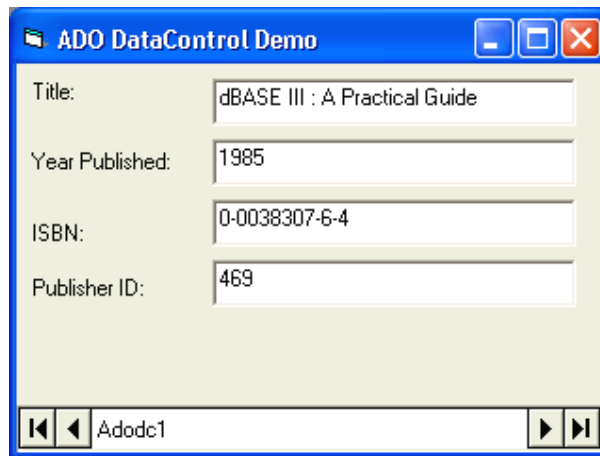
Bây giờ chúng ta hãy đặt lên Form 4 labels với captions: Title, Year Published, ISBN và Publisher ID. Kế đó cho thêm 4 textboxes tương ứng và đặt tên chúng là txtTitle, txtYearPublished, txtISBN và txtPublisherID.

Để thực hiện Data Binding, chúng ta hãy chọn textbox txtYearPublished (năm xuất bản), rồi set property Datasource của nó trong Properties Window thành Adodc1. Khi click lên property DataField của txtYearPublished và mở ComboBox ra chúng ta sẽ thấy liệt kê tên các Trường trong bảng Titles. Đó là vì Adodc1 được coi như trung gian lấy bảng Titles từ cơ sở dữ liệu. Ở đây ta sẽ chọn cột Year Published.

Lập lại công tác này cho 3 textboxes kia, và chọn các cột Title (Tiêu đề), ISBN (số lý lịch trong thư viện quốc tế), và PubID (số lý lịch nhà xuất bản) làm DataField cho chúng.



Đến đây, mặc dầu chưa viết một dòng code nào, chúng ta có thể chạy chương trình và nó sẽ hiển thị như dưới đây:

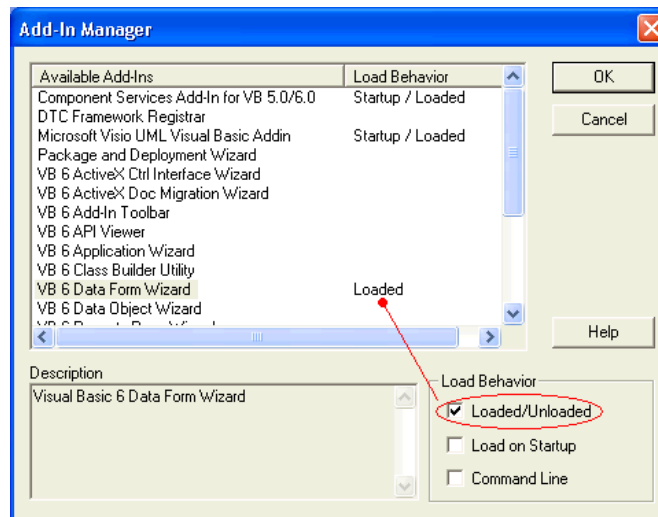


#### IV. Data Form Wizard

Để giúp lập trình viên thiết kế các data forms nhanh hơn, VB6 cho ta Data Form Wizard để generate (phát sinh) ra một form có hỗ trợ Edit, Add và Delete bản ghi.

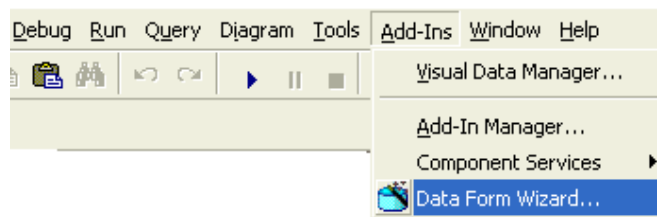
Bây giờ chúng ta hãy khởi động một standard project VB6 mới, tên ADOClass và copy MS Access file BIBLIO.MDB, tức là cơ sở dữ liệu, vào trong cùng thư mục của dự án mới này.

Muốn dùng Data Form Wizard, trước hết ta phải thêm nó vào môi trường phát triển (IDE) của VB6. Chúng ta hãy dùng IDE Menu Command Add-Ins | Add-In Manager.... Chọn VB6 Data Form Wizard trong giao thoại, rồi click Checkbox Loaded/Unloaded để chữ Loaded hiện bên phải dòng "VB6 Data Form Wizard" như trong hình dưới đây:

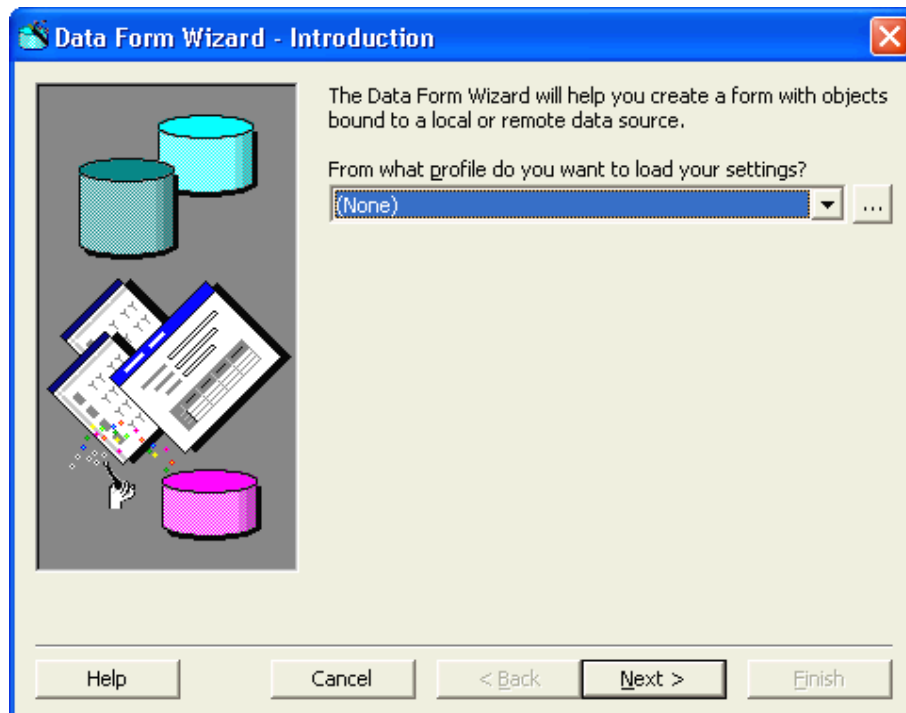


Nếu chúng ta muốn mỗi lần khởi động VB6 IDE là có sẵn Data Form Wizard trong menu Add-Ins thì ngoài option Loaded, chúng ta click thêm check box Load on Startup.

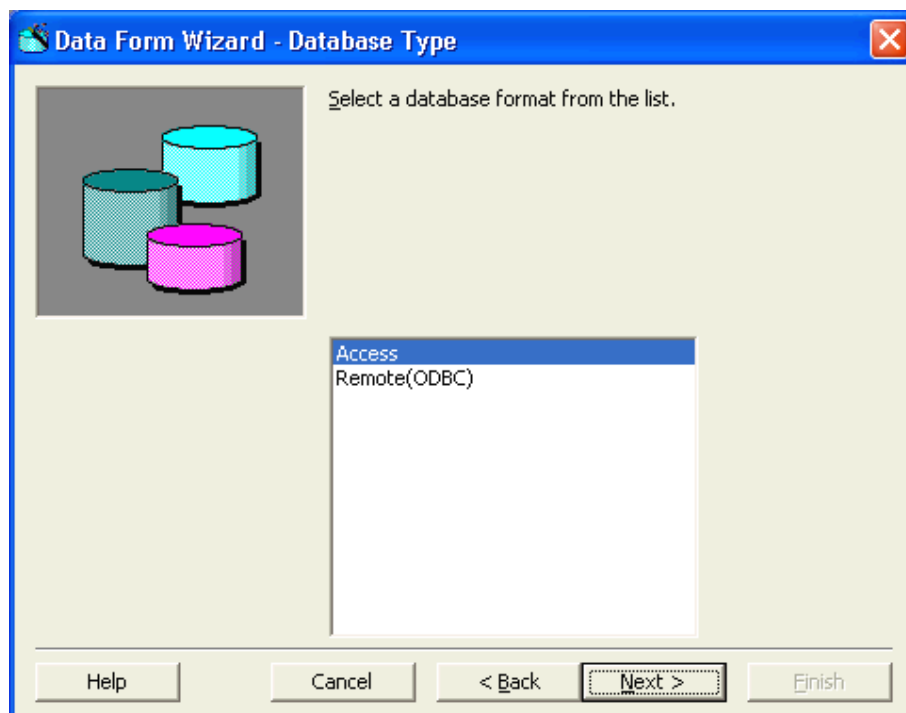
Một Add-In là một menu Item mới mà ta có thể thêm vào một chương trình ứng dụng có sẵn. Thường thường, người ta dùng Add-Ins để thêm chức năng cho một chương trình, làm như là chương trình đã có sẵn chức năng ấy từ đầu. Chúng ta hãy khởi động Data Form Wizard từ IDE Menu Command mới Add-Ins | Data Form Wizard...



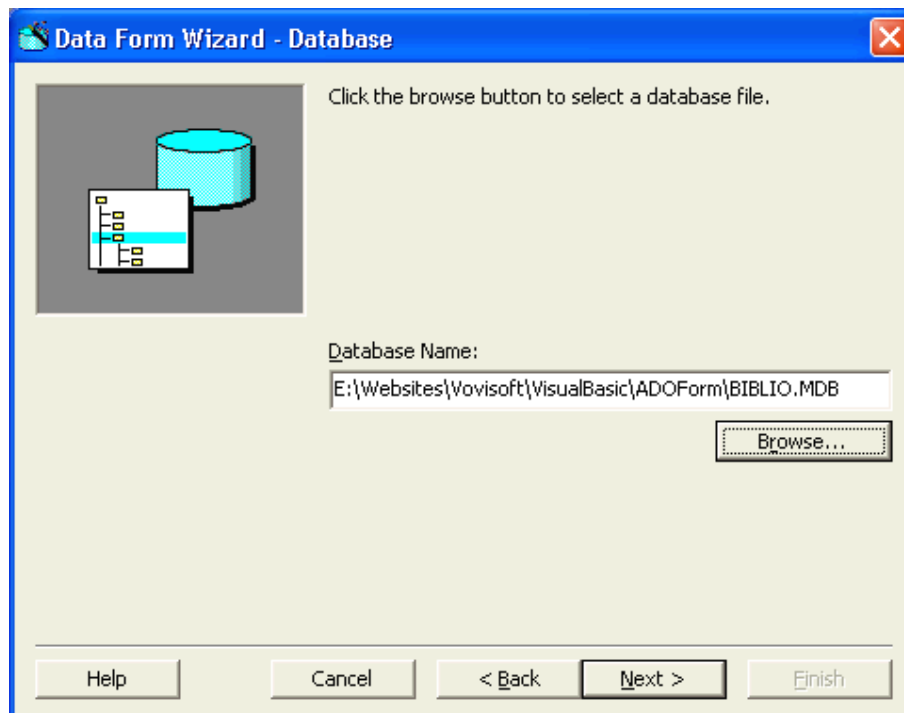
Khi trang Data Form Wizard - Introduction hiện ra, click Next



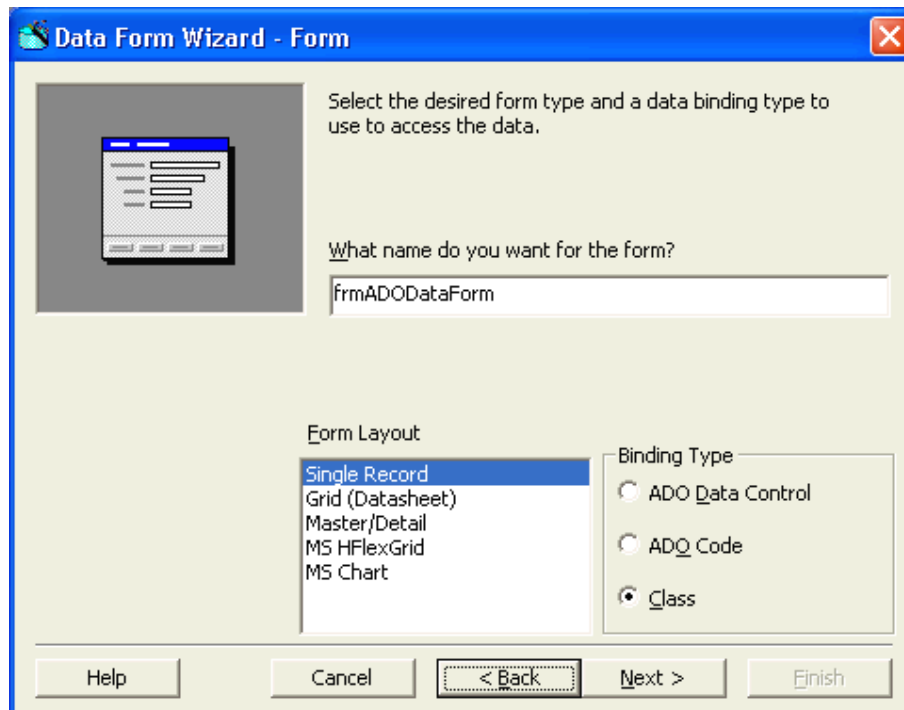
Trong trang kế đó chọn Access làm Database Type.



Trong trang Database, click Browse để chọn một MS Access database file. Ở đây ta chọn file BIBLIO.MDB từ chính thư mục của chương trình này. Đoạn click Next.



Trong trang Form, ta chọn Single Record cho Form Layout và Class cho Binding Type. Đoạn click Next. Nếu ta chọn ADO Data Control thì kết quả sẽ giống giống như khi ta dùng Control Data DAO như trong một phần trước.

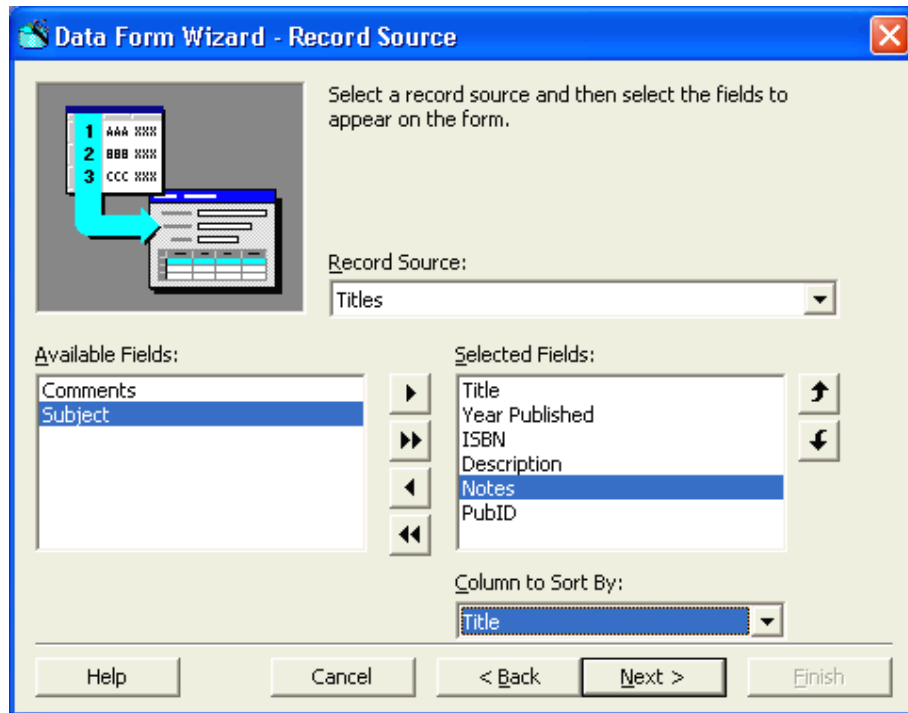


Trong trang bản ghi Source ta chọn bảng Titles. Listbox của Available Fields sẽ hiển thị các trường của bảng Titles. Sau khi chọn một trường bằng cách click lên tên trường ấy trong Listbox, nếu chúng ta click hình tam giác chỉ qua phải thì tên trường ấy sẽ được dời qua nằm dưới cùng trong Listbox Selected Fields bên phải.



Nếu chúng ta click hình hai tam giác chỉ qua bên phải thì tất cả mọi trường còn lại bên trái sẽ được dời qua bên phải. Chúng ta cũng có thể sắp đặt vị trí của các trường đã chọn bằng cách click lên tên trường ấy rồi click hình mũi tên chỉ lên hay xuống để di chuyển trường ấy lên hay xuống trong danh sách các trường.

Ngoài ra, chúng ta hãy chọn Title làm Column to Sort By trong cái Combobox của nó để các bản ghi trong Recordset được sắp xếp theo thứ tự ABC (alphabetical order) của trường Tiêu đề (Title).



Trong trang Control Selection, ta sẽ để y nguyên để có đủ mọi buttons. Chúng ta hãy click Next.



Khi Data Form Wizard chấm dứt, nó sẽ generate form frmADODataForm. Chúng ta hãy remove Form1 và dùng Menu Command Project | ADODataControl Properties... để đổi Startup Object thành frmADODataForm. Thế là tạm xong chương trình để Edit các bản ghi của bảng Titles.

Chúng ta hãy quan sát cái Form và phần code được Data Form Wizard generated. Trong frmADODataForm, các textboxes làm thành một array tên txtFields. Mọi textbox đều có property DataField định sẵn tên trường của bảng Titles. Ví dụ như txtFields(2) có DataField là ISBN. Form chính không dùng Control Data ADO nhưng dùng một Object của class clsTitles.

Phần Initialisation của class clsTitles là Open một Connection và lấy về một Dataset có tên DataMember là Primary như sau:

```
Private Sub Class_Initialize()  
    Dim db As Connection  
    Set db = New Connection  
    db.CursorLocation = adUseClient  
    ' Open connection  
    db.Open "PROVIDER=Microsoft.Jet.OLEDB.3.51;Data  
Source=E:\Websites\Vovisoft\VisualBasic\ADODForm\BIBLIO.MDB;"  
    ' Instantiate ADO recordset  
    Set adoPrimaryRS = New Recordset  
    ' Retrieve data for Recordset  
    adoPrimaryRS.Open "select Title,[Year  
Published],ISBN,Description,Notes,PubID from Titles Order by Title",  
_ db, adOpenStatic, adLockOptimistic  
    ' Define the only data member, named Primary  
    DataMembers.Add "Primary"  
End Sub
```

Về vị trí của cơ sở dữ liệu, nếu chúng ta không muốn ở một thư mục nào thì dùng App.Path để xác định mối liên hệ giữa vị trí của cơ sở dữ liệu và thư mục của chính chương trình đang chạy, ví dụ như:

```
db.Open "PROVIDER=Microsoft.Jet.OLEDB.3.51;Data Source=" &  
App.Path & "\BIBLIO.MDB;"
```

Trong Sub Form\_Load, ta có thể dùng For Each để đi qua hết các textboxes trong array txtFields. Vì property Datasource của textbox là một Object nên ta dùng keyword Set để point nó đến Object PrimaryCLS. Đồng thời ta cũng phải chỉ định tên của DataMember của mỗi textbox là Primary:

```
Private Sub Form_Load()
```

```

    ' Instantiate an Object of class clsTitles
    Set PrimaryCLS = New clsTitles
    Dim oText As TextBox
    ' Iterate through each textbox in the array txtFields
    ' Bind the text boxes to the data source, i.e. PrimaryCLS
    For Each oText In Me.txtFields
        oText.DataMember = "Primary"
        ' Use Set because property Datasource is an Object
        Set oText.DataSource = PrimaryCLS
    Next
End Sub

```

Khi sự di chuyển từ bản ghi này đến bản ghi khác chấm dứt, chính Recordset có raise Event MoveComplete. Event ấy được handled (giải quyết) trong class clsTitles bằng cách lại raise Event MoveComplete để nó được handled trong Form.

```

Dim WithEvents adoPrimaryRS As Recordset
Private Sub adoPrimaryRS_MoveComplete(ByVal adReason As
ADODB.EventReasonEnum, _
    ByVal pError As ADODB.Error, adStatus As
ADODB.EventStatusEnum, ByVal pRecordset As ADODB.Recordset)
    ' Raise event to be handled by main form
    RaiseEvent MoveComplete
End Sub

```

Muốn handle Event trong clsTitles ta phải declare recordset adoPrimaryRS với WithEvents:

Và trong Form ta cũng phải declare (object clsTitles) PrimaryCLS với WithEvents:

```
Private WithEvents PrimaryCLS As clsTitles
```

Trong Form, Event MoveComplete sẽ làm hiển thị vị trí tuyệt đối (Absolute Position) của bản ghi bằng code dưới đây:

```

Private Sub PrimaryCLS_MoveComplete()
    'This will hiển thị the current record position for this
    recordset
    lblStatus.Caption="Record: " & CStr(PrimaryCLS.AbsolutePosition)
End Sub

```

Khi người sử dụng nhấn Refresh, các textboxes sẽ được hiển thị lại với chi tiết mới nhất của bản ghi từ trong recordset, nhờ khi có ai khác đã sửa đổi bản ghi. Phương thức Requery của clsTitles lại gọi phương thức Requery của Recordset như sau:

```

Private Sub cmdRefresh_Click()
    'This is only needed for multi user applications
    On Error GoTo RefreshErr
    ' fetch the latest copy of Recordset
    PrimaryCLS.Requery
    Exit Sub
RefreshErr:
    MsgBox Err.Description
End Sub

'In Class clsTitles
Public Sub Requery()
    ' Fetch latest copy of record
    adoPrimaryRS.Requery

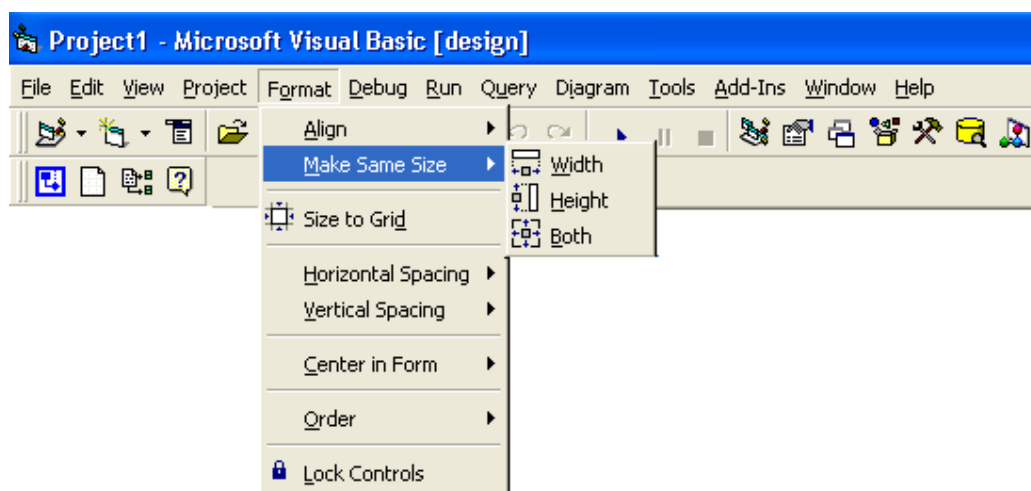
```

```
DataMemberChanged "Primary"  
End Sub
```

## CHƯƠNG 9. THIẾT KẾ HỆ THỐNG THỰC ĐƠN

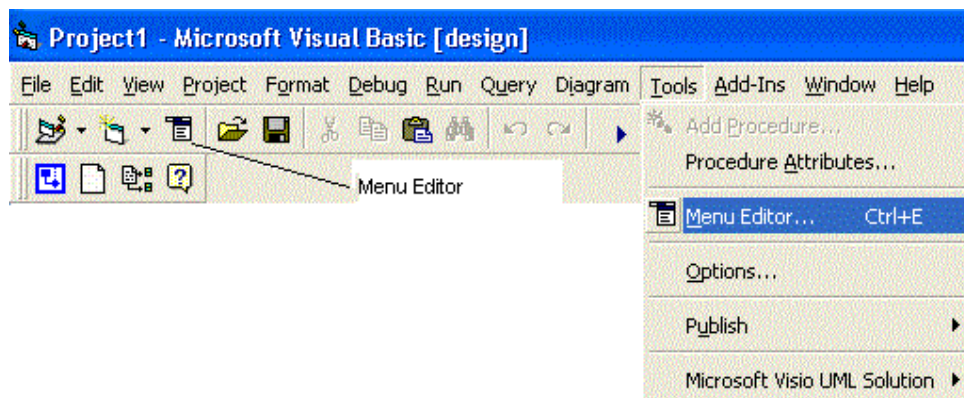
Thực đơn (menu) trong Windows là nơi tất cả các chức năng của một chương trình được sắp xếp thứ tự theo từng loại để giúp người sử dụng có thể gọi thực hiện một cách dễ dàng.

Có hai loại menu ta thường gặp : Menu Bar (thanh menu ngang) và Pop-Up Menu (thực đơn chỉ xuất hiện khi được chọn). Ta thường dùng Menu Bar là thực đơn chính và thường đặt nó nằm ở phía trên đỉnh màn hình. Nằm dọc theo chiều ngang của Menu Bar là tên các nhóm chức năng chính, nếu ta click lên một chức năng nào đó trong Menu Bar thì chương trình sẽ thả xuống một Pop-Up với những MenuItem nằm dọc theo chiều thẳng đứng. Nếu ta click lên MenuItem nào có dấu hình tam giác nhỏ bên phải thì chương trình sẽ popup một Menu như trong hình dưới đây (khi ta click **Format** | **Make Same Size**):

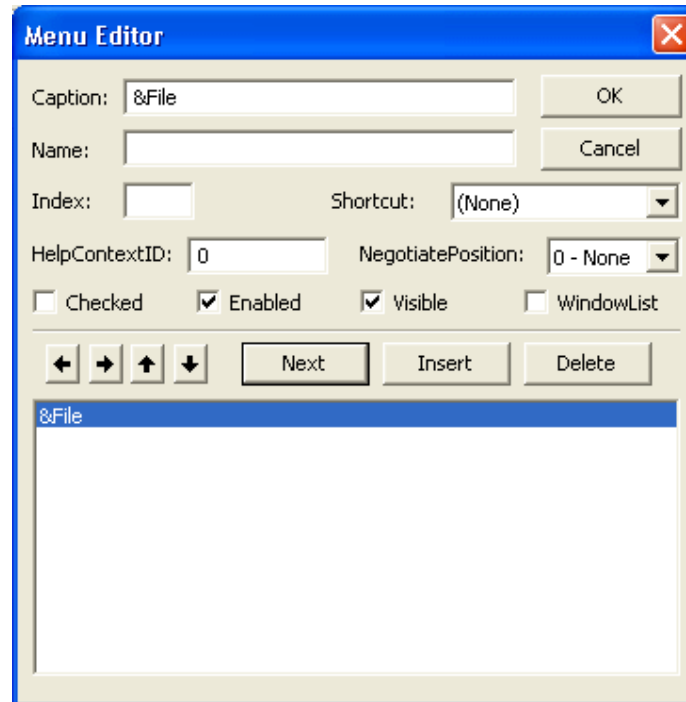


### I. Main Menu (Thực đơn chính)

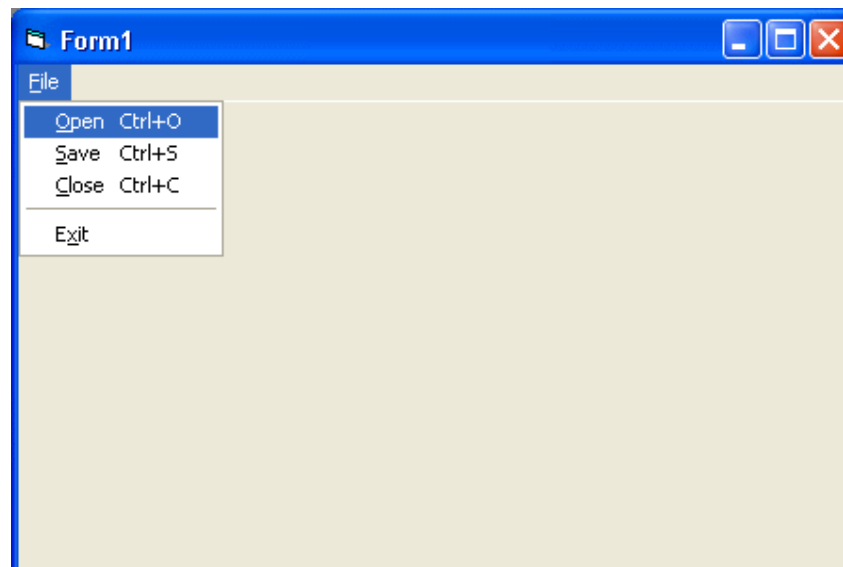
Ta dùng **Menu Editor** để tạo hoặc sửa một thực đơn cho chương trình. Thực đơn thuộc về một Form. Do đó, trước hết ta chọn một Form để làm việc với Designer của nó. Kế đó ta dùng Menu Command **Tools** | **Menu Editor** hay click lên biểu tượng của Menu Editor trên Toolbar để làm cho Menu Editor hiện ra.



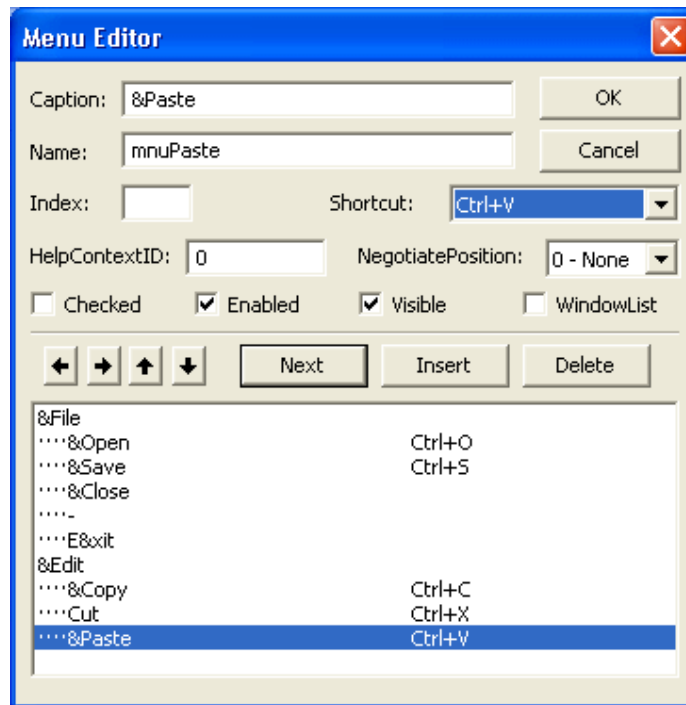
Đầu tiên có một vệt màu xanh nằm trong khung trắng của Menu Editor, nơi sẽ hiển thị Caption của Menu Command đầu tiên của Form. Khi ta đánh chữ **&File** vào Textbox **Caption**, nó cũng hiện ra trên vệt xanh nói trên. Kế đó, chúng ta có thể đánh tên của Menu Command vào Textbox **Name**. Dù ta cho Menu Command một tên nhưng ta ít khi dùng nó, trừ trường hợp muốn nó visible/invisible (hiện ra/biến mất). Bình thường ta dùng tên của MenuItems nhiều hơn.



Để có một Menu như trong hình dưới đây ta còn phải hiệu chỉnh thêm vào các MenuItems Open, Save, Close và Exit.



Hình dưới đây cho thấy tất cả các MenuItems của Menu Command File đều nằm thụt qua bên phải với bốn dấu chấm (....) ở phía trước. Khi ta click dấu tên chỉ qua phải thì MenuItem ta đang hiệu chỉnh sẽ có thêm bốn dấu chấm, tức là thụt một bậc trong Menu (Nested).



Tương tự như vậy, khi ta click dấu tên chỉ qua trái thì MenuItem ta đang Edit sẽ mất bốn dấu chấm, tức là trôi một bậc trong Menu.

Nếu muốn cho người sử dụng dùng Alt key để xử dụng Menu, chúng ta đánh thêm dấu **&** trước ký tự chúng ta muốn trong nhãn của thực đơn. Ví dụ **Alt-F** sẽ thả xuống Menu của Menu Command File.

Nếu chúng ta đặt cho MenuItem **&Open** tên **mnuOpen**, thì khi chúng ta Click lên Caption nó trên Form trong lúc thiết kế, VB6 IDE sẽ hiển thị cái vỏ của **Sub mnuOpen\_Click()**, giống như Sub cmdButton\_Click() của một CommandButton:

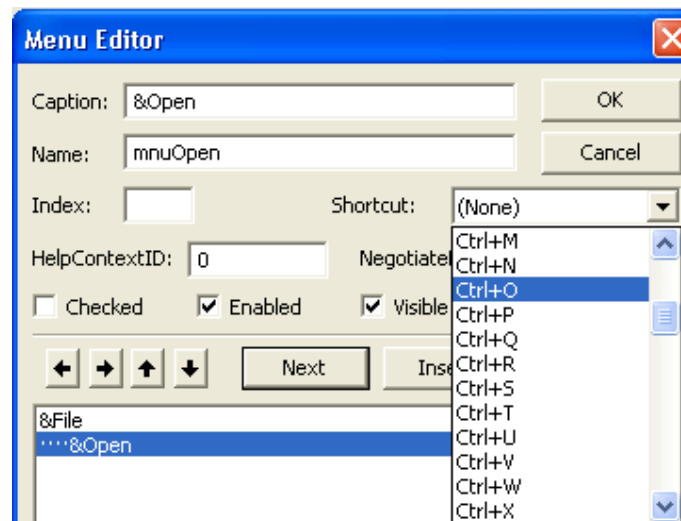
```
Private Sub mnuOpen_Click()
    MsgBox "You clicked mnuOpen"
End Sub
```

Trong ví dụ trên ta đánh thêm một chỉ thị để hiển thị một thông điệp đơn giản **"You clicked mnuOpen"**. Chúng ta có thể đặt cho một MenuItem tên gì cũng được, nhưng người ta thường dùng tiền vị từ **mnu** để dễ phân biệt một menuItem Event với một CommandButton Event. Do đó, ta có những tên mnuFile, mnuOpen, mnuSave, mnuClose, mnuExit.

Dấu gạch ngang giữa MenuItems Close và Exit được gọi là **Menu Separator**. Chúng ta có thể chèn một dấu ngăn cách Menu bằng cách cho Caption nó bằng dấu trừ ( - ).

Ngoài Alt key ta còn có thể cho người sử dụng dùng Shortcut của menuItem. Để cho MenuItem một Shortcut, chúng ta chọn cho nó một Shortcut từ ComboBox **Shortcut** trong Menu Editor.

Trong hình dưới đây ta chọn **Ctrl+O** cho mnuOpen.



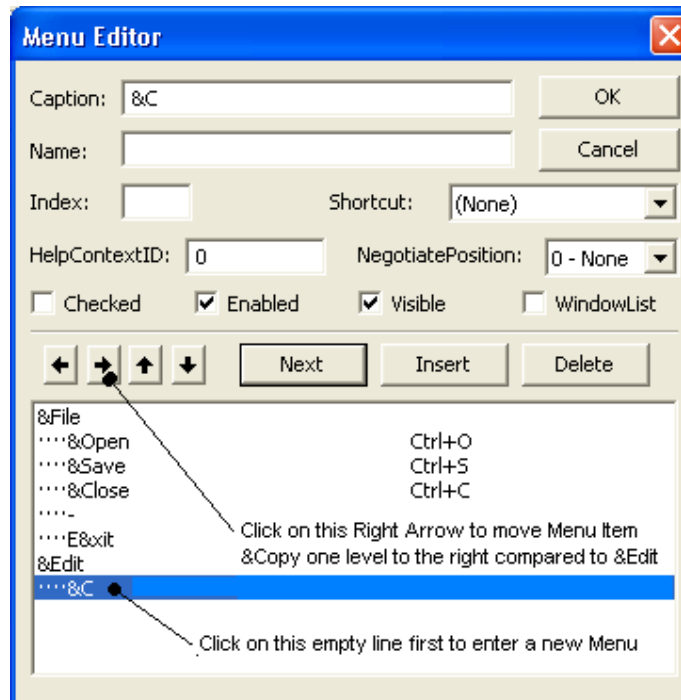
Ở chế độ mặc định, menuItem được Enabled và Visible. Lúc thiết kế chúng ta có thể cho MenuItem giá trị khởi đầu của Enabled và Visible bằng cách dùng Checkboxes Enabled và Visible.

Trong khi chạy chương trình (at runtime), chúng ta cũng có thể thay đổi các values Enabled và Visible như sau:

```
mnuSave.Enabled = False
mnuOpen.Visible = False
```

Khi một MenuItem có Enabled=False thì nó bị mờ và người sử dụng không dùng được.

Chúng ta dùng các dấu mũi tên chỉ lên và xuống để di chuyển MenuItem đã được lựa chọn lên và xuống trong danh sách các MenuItems. Chúng ta dùng nút **Delete** để hủy bỏ MenuItem đã được lựa chọn, **Insert** để chèn một MenuItem mới ngay trên MenuItem đã được lựa chọn và **Next** để chọn MenuItem ngay dưới MenuItem đã được lựa chọn.

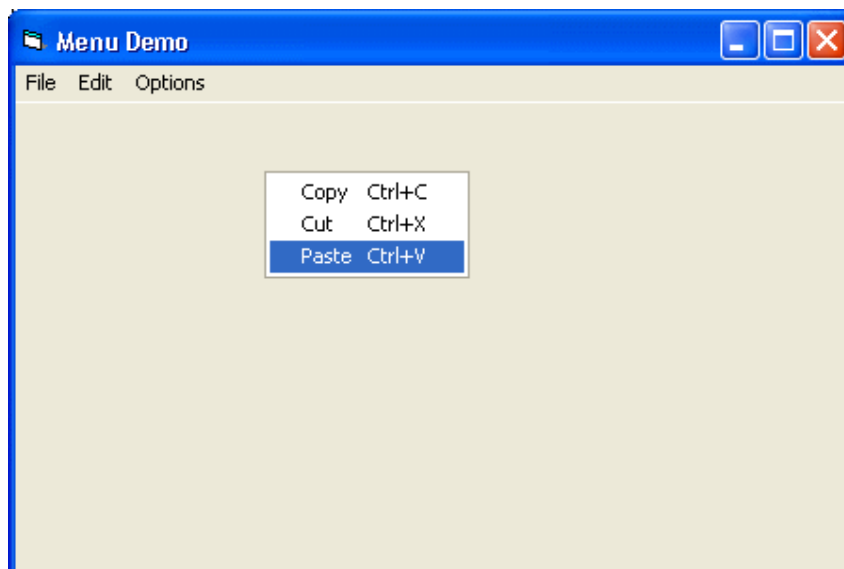




## II. Pop-up Menu

Đối với người sử dụng, đang khi làm việc với một Object trong Windows tiện nhất là ta có thể làm hiển thị Context Menu (menu áp dụng theo tình huống) bằng một nhấp chuột. Thông thường đó là Right Click (nhấn phím phải của chuột) và Context Menu còn được gọi là Pop-up Menu. Pop-Up menu thật ra là Drop-down menu của một Menu Bar Command. Bình thường Menu Bar Command ấy có thể visible (nhìn thấy) hay invisible (không nhìn thấy).

Trong hình dưới đây, khi người sử dụng Right click trên Form, mnuEdit sẽ hiện lên. Nếu bình thường chúng ta không muốn cho người sử dụng dùng nó trong Main Menu thì chúng ta cho nó invisible:



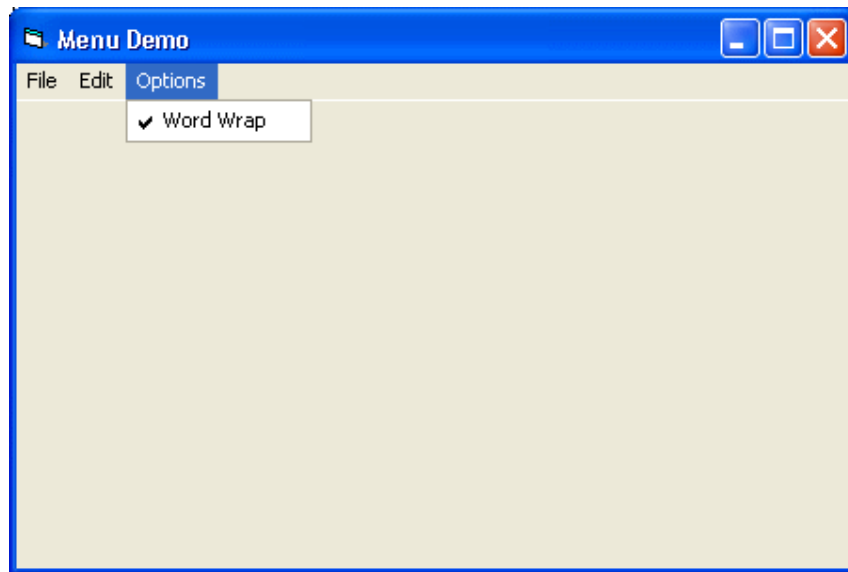
Code làm cho Popup menu hiện lên được viết trong Event Mousedown của một Object mà tình cờ ở đây là của chính cái Form:

```
Private Sub Form_MouseDown(Button As Integer, Shift As Integer, X As Single, Y As Single)
    ' Popup the Edit Menu if User clicked the Right Button of the Mouse
    If Button = vbRightButton Then
        PopupMenu mnuEdit
    End If
End Sub
```

Ngay cả khi chúng ta muốn cho mnuEdit bình thường là invisible, chúng ta cũng nên để cho nó visible trong lúc đầu để tiện bỏ code vào dùng để xử lý Click Events của những MenuItems thuộc về mnuEdit như mnuCopy, mnuCut và mnuPaste.

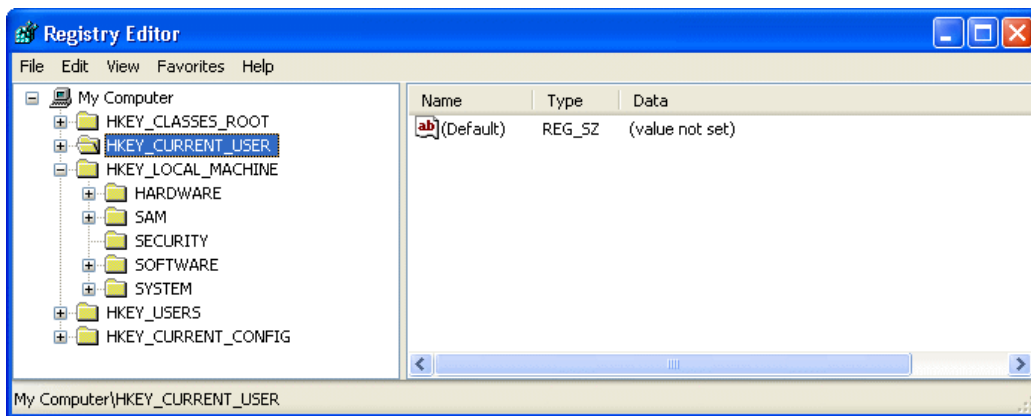
## III. Lưu trữ việc thiết lập thực đơn trong Registry

Giả sử chương trình chúng ta cho người sử dụng một Option WordWrap như dưới đây:



Chúng ta muốn chương trình nhớ Option mà người sử dụng đã chọn, để lần tới khi người sử dụng khởi động chương trình thì Option WordWrap còn giữ nguyên giá trị như cũ.

Cách tiện nhất là chứa giá trị của Option WordWrap như một **Key** trong **Registry**. Registry là một loại cơ sở dữ liệu đặc biệt của hệ điều hành Windows dùng để chứa những dữ kiện liên quan đến Users, Hardware, Configurations, ActiveX Components... dùng trong máy tính. Trong Registry, dữ liệu được sắp đặt theo từng loại. Chúng ta có thể sửa đổi trực tiếp trị số các Keys trong Registry bằng cách dùng **Registry Editor**.



Trong chương trình này ta cũng lập trình để chương trình nhớ luôn vị trí của Form khi chương trình ngừng lại, để lần tới khi người sử dụng khởi động chương trình thì chương trình sẽ có vị trí lúc đầu giống y như trước.

Ta sẽ dùng **Sub SaveSetting** để chứa giá trị **Checked** của mnuWordWrap và **Left, Top** của Form. Chương trình này ta sẽ để trong **Sub Form\_QueryUnload** vì nó sẽ được thực thi trước khi Form Unload.

```
Private Sub Form_QueryUnload(Cancel As Integer, UnloadMode As Integer)
    SaveSettings
End Sub

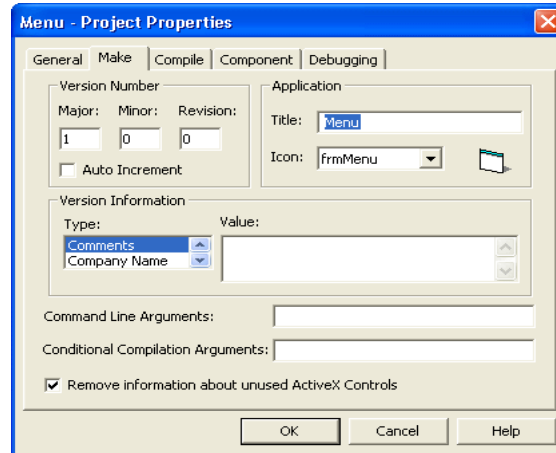
Private Sub SaveSettings()
    ' Save Location of the form
    SaveSetting App.Title, "Location", "Left", Me.Left
```

```

SaveSetting App.Title, "Location", "Top", Me.Top
' Save the setting of WordWrap in menu
SaveSetting App.Title, "Settings", "WordWrap",
mnuWordWrap.Checked
End Sub

```

App.Title là tiêu đề của chương trình. Thông thường nó là tên của VB Project, nhưng chúng ta có thể sửa nó trong **Project Property Dialog (Tab Make)** :



Khi chứa giá trị của một **Key** vào Registry chúng ta có thể sắp đặt cho nó nằm trong **Section** nào tùy ý. Ở đây ta đặt ra hai Sections tên **Location** để chứa Top, Left của Form và tên **Settings** để chứa Key mnuWordWrap.Checked.

Muốn cho chương trình có các giá trị của Keys chứa trong Registry khi nó khởi động ta chỉ cần dùng **Function GetSetting** trong **Sub Form\_Load** để đọc vào từ Registry như dưới đây:

```

Private Sub Form_Load()
' Initialise Location of the form by reading the Settings from
the Registry
Me.Left = Val(GetSetting(App.Title, "Location", "Left", "0"))
Me.Top = Val(GetSetting(App.Title, "Location", "Top", "0"))
' Initialise setting of WordWrap in the menu
mnuWordWrap.Checked = ( GetSetting(App.Title, "Settings",
"WordWrap", "False") = "True" )
End Sub

```

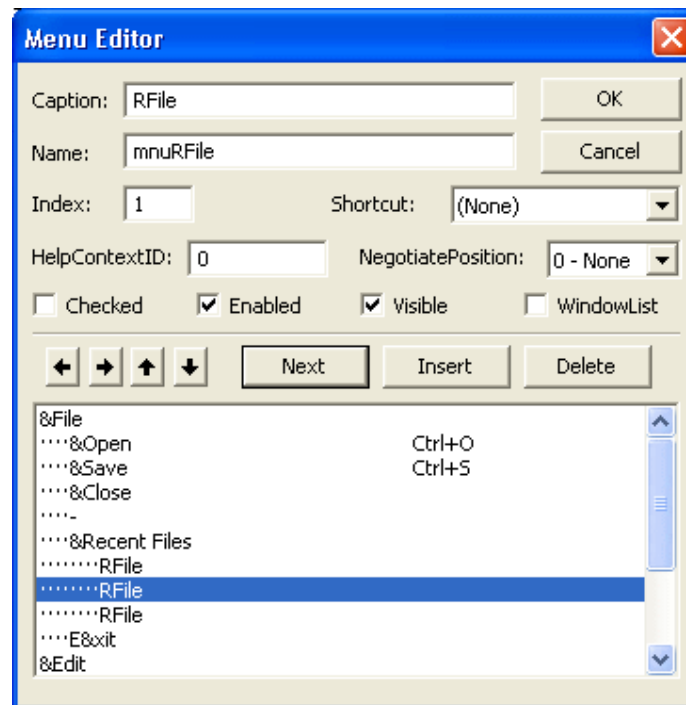
Lúc đầu khi chưa có gì trong Registry thì **"0"** (string "0" đã được đổi từ kiểu số sang) là giá trị mặc định cho Left và Top, còn **"False"** là giá trị mặc định của mnuWordWrap.Checked.

Ngoài ra, ta cũng muốn chương trình nhớ tên của 3 tập tin người dùng sử dụng gần đây nhất. Tức là trong Drop-down của Menu Command File sẽ có MenuItem **Recent Files** để hiển thị từ một đến ba tên tập tin, cái mới nhất nằm ở trên cùng. Trước hết, ta cần tạo ra 3 SubmenuItem có cùng tên mnuRFile nhưng mang **Index** bằng 0, 1 và 2 (chúng ta đánh vào Textbox **Index**). Ta sẽ dùng Captions của chúng để hiển thị tên các tập tin. Lúc chưa có Filename nào cả thì MenuItem **Recent Files** sẽ bị làm mờ đi (tức là mnuRecentFiles.Enabled = False ).

Ta sẽ chứa tên các tập tin như một xâu ký tự trong Section Settings của Registry. Ta phân cách tên các tập tin bằng ký tự đặc biệt "|".

Ví dụ: "LattestFileName.txt|OldFileName.txt|OldestFileName.txt"

Mỗi lần người sử dụng mở một tập tin ta sẽ thêm tên tập tin ấy vào trong Registry và bất cứ lúc nào chỉ giữ lại tên của 3 tập tin mới dùng nhất.

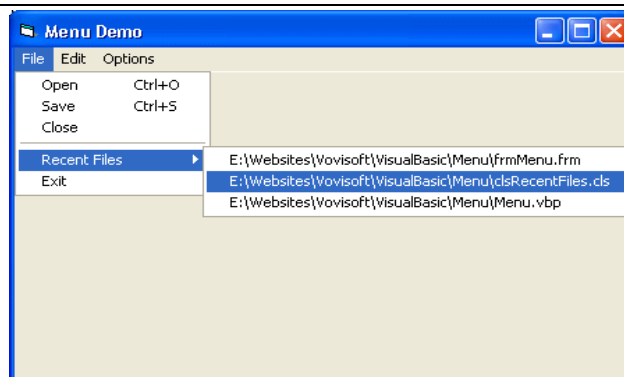


Dưới đây là đoạn chương trình dùng để thêm tên tập tin mới dùng nhất vào Registry:

```
Private Sub mnuOpen_Click()
    ' Initialise Folder in Common Dialog
    CommonDialog1.InitDir = App.Path
    ' Launch the dialog
    CommonDialog1.ShowOpen
    ' Save the Filename in the Registry, using Object
    myRecentFiles
    myRecentFiles.AddFile CommonDialog1.FileName
End Sub
```

Và chương trình dùng trong Sub Form\_Load để đọc tên RecentFiles và hiển thị trong Menu:

```
Set myRecentFiles = New clsRecentFiles
' Pass the form handle to it
' This effectively loads the most recently used FileNames to menu
myRecentFiles.Init Me
```



Ta sẽ dùng một Class tên **clsRecentFiles** để đặc biệt lo việc chứa tên các tập tin vào Registry và hiển thị tên các tập tin ấy trong thực đơn. Bên trong clsRecentFiles ta cũng dùng [clsString](#), là một Class giúp ta ngắt đoạn xâu ký tự trong Registry ra tên của các tập tin dựa vào chỗ các ký tự phân cách |.

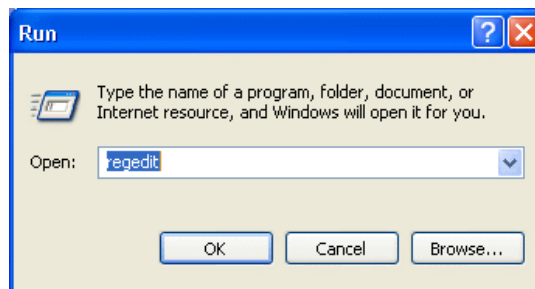
```
' Class Name: clsRecentFiles
' This Class saves the most Recent FileNames used in the Registry
' in form of a String delimited by |.
' Up to MaxFiles Filenames maybe stored.
' You need to pass the Form that contains the menu to it.
' The assumption is that you have created an array of MenuItems
' named mnuRFile to display the FileNames
Const MaxFiles = 3 ' Maximum number of FileNames to remember
Private myForm As Form
Private RecentFiles As clsString
Public Sub Init(TForm As frmMenu)
    Set myForm = TForm
    Set RecentFiles = New clsString
    ' Read the Most Recent Filename String from the Registry
    RecentFiles.Text = GetSetting(App.Title, "Settings",
"RecentFiles", "")
    ' Assign the Delimiter character and tokennise the String
    (i.e. split it) into FileNames
    RecentFiles.Delimiter = "|"
    UpdateMenu
End Sub
Public Sub AddFile(Filename As String)
    ' Add the latest FileName to the list and update the Registry
    ' Prefix the FileName to the existing MostRecentFileName
String
    RecentFiles.Text = Filename & "|" & RecentFiles.Text
    ' Discard the oldest FileNames if the total number is greater
than MaxFiles
    If RecentFiles.TokenCount > MaxFiles Then
        Dim TStr As String
        Dim i As Integer
        ' Reconstitute the String that contains only the most
recent MaxFiles FileNames
        For i = 1 To MaxFiles
            TStr = TStr & RecentFiles.TokenAt(i) & "|"
        Next
        ' Remove the last delimiter character on the right
        RecentFiles.Text = Left(TStr, Len(TStr) - 1)
    End If
    ' Update the String in the Registry
    SaveSetting App.Title, "Settings", "RecentFiles",
RecentFiles.Text
    UpdateMenu
End Sub
Private Sub UpdateMenu()
    ' Hiển thị the most recent Filenames in the menu
    Dim i As Integer
```

```

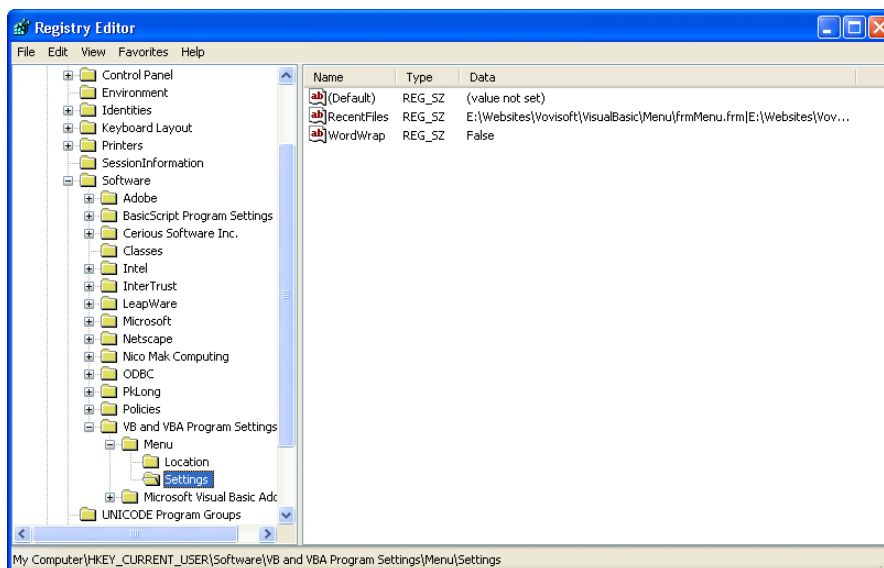
' If there is no FileNames to hiển thị then disable the
MenuItem entry
If RecentFiles.TokenCount = 0 Then
    myForm.mnuRecentFiles.Enabled = False
    Exit Sub
Else
    ' Otherwise enable the MenuItem entry
    myForm.mnuRecentFiles.Enabled = True
End If
' Assign FileName to Caption of mnuRFile array
' and make the MenuItem elements visible
For i = 1 To RecentFiles.TokenCount
    myForm.mnuRFile(i - 1).Caption = RecentFiles.TokenAt(i)
    myForm.mnuRFile(i - 1).Visible = True ' Make the MenuItem
If i = MaxFiles Then Exit
Next
' Make the rest of the MenuItem array mnuRFile invisible
If RecentFiles.TokenCount < MaxFiles Then
    For i = RecentFiles.TokenCount To MaxFiles - 1
        myForm.mnuRFile(i).Visible = False
    Next
End If
End Sub

```

Chúng ta có thể chạy dòng lệnh **RegEdit** sau khi chọn **Start | Run**



để xem chi tiết của các Keys mà chương trình đã chứa trong Sections **Location** và **Settings** của Thư mục **HKEY\_CURRENT\_USER\Software\VB and VBA Program Settings\Menu**



## CHƯƠNG 10. GỠ LỖI CHƯƠNG TRÌNH

Bugs là những lỗi của chương trình mà ta phát hiện khi chạy nó. Debug là công việc loại tất cả những lỗi trong chương trình để nó chạy tốt trong mọi tình huống.

Thông thường muốn sửa một lỗi nào đó trước hết ta phải tìm hiểu lý do khiến nó xuất hiện. Một khi đã biết được nguyên nhân rồi ta sẽ nghĩ ra cách giải quyết. Nói chung, có hai loại bugs : hoặc là chương trình không làm đúng chuyện cần phải làm vì lập trình viên hiểu sai yêu cầu/đặc tả hay được cho tin tức sai lạc, hoặc là chương trình bỏ sót chi tiết cần phải xử lý. Trường hợp này ta giải quyết bằng cách giảm thiểu sự hiểu lầm qua sự nâng cấp khả năng truyền thông.

Chương trình không thực hiện đúng như ý lập trình viên muốn, tức là lập trình viên muốn một đằng mà bảo chương trình làm một ngã vì vô tình không viết chương trình đúng cách. Trường hợp này ta giải quyết bằng cách dùng những công cụ phần mềm (kể cả ngôn ngữ lập trình) thích hợp, và có những quá trình làm việc có hệ thống.

Có nhiều yếu tố ảnh hưởng đến chất lượng của một chương trình như chức năng của chương trình, cấu trúc của các bộ phận, kỹ thuật lập trình và phương pháp debug. Debug không hẳn nằm ở giai đoạn cuối của dự án mà tùy thuộc rất nhiều vào các yếu tố kể trên trong mọi giai đoạn triển khai.

### I. Đặc tả chương trình (Program Specifications)

Dầu chương trình lớn hay nhỏ, trước hết ta phải xác định rõ ràng và tỉ mỉ nó cần phải làm gì, bao nhiêu người dùng, mạng như thế nào, cơ sở dữ liệu lớn bao nhiêu, phải chạy nhanh đến mức nào...

Có nhiều chương trình phải bị thay đổi nữa chừng vì lập trình viên hiểu lầm điều khách hàng muốn. Do đó, khi triển khai một dự án phần mềm ta cần phải áp dụng chặt chẽ qui trình phát triển phần mềm (xem trong môn học Công nghệ phần mềm hoặc đọc thêm các tài liệu về Software Engineering).

#### 1. Cấu trúc các bộ phận

Chương trình nào cũng có một kiến trúc tương tự như một cỗ máy. Mỗi bộ phận càng đơn giản càng tốt và cách ráp các bộ phận phải như thế nào để ta dễ dùng thử, dễ kiểm tra (testing). Trong khi thiết kế ta phải biết trước những yếu điểm của mỗi bộ phận nằm ở đâu để ta chuẩn bị cách thử chúng. Ta sẽ không thể tin bộ phận nào hoàn hảo cho đến khi đã thử nó, dù nó đơn giản đến đâu.

Nếu ta muốn dùng một kỹ thuật gì trong một hoàn cảnh nào mà ta không biết chắc nó chạy không thì nên thử riêng rẽ nó trước. Phương pháp ấy được gọi là **Prototype**.

Ngoài ra, ta cũng nên xây dựng những kịch bản kiểm thử cho những trường hợp đặc biệt, điển hình là dùng bad data (dữ liệu xấu) - khi người sử dụng bấm lung tung hay cơ sở dữ liệu chứa nhiều rác.

Nếu chương trình chạy trong **real-time** (tức là dữ liệu thu nhập qua cổng COM, hay qua mạng), chúng ta cần phải lưu ý những trường hợp khác nhau tùy theo việc gì xảy ra trước, việc

gì xảy ra sau. Phương pháp kiểm thử tốt nhất là xây dựng trước những kịch bản và dữ liệu để có thể thử từng giai đoạn và tình huống.

Ngày nay với kỹ thuật hướng đối tượng, ở giai đoạn thiết kế này là lúc quyết định các cấu trúc dữ liệu (bảng, mảng, bản ghi...) và số lượng các Form, Class. Lưu ý rằng, mỗi Class gồm có một cấu trúc dữ liệu và những Subs/Functions/Properties làm việc trên dữ liệu ấy. Cấu trúc dữ liệu phải chứa đầy đủ những chi tiết ta cần (tên trường, tên kiểu dữ liệu...). Kế đó là những cách chương trình xử lý dữ liệu. Subs/Functions nào có thể cho bên ngoài gọi thì ta cho nó **Public**, còn những Subs/Functions khác hiện hữu để phục vụ bên trong class thì ta cho nó **Private**.

## 2. Kỹ thuật lập trình

Kiến thức cơ bản của lập trình viên và các thói quen của họ rất quan trọng. Nói chung, những người hấp tấp, nhảy vào viết chương trình trước khi suy nghĩ hay cân nhắc chín chắn thì sau này bugs xuất hiện nhiều là điều tự nhiên.

## 3. Dùng Subs và Functions

Nếu ở giai đoạn thiết kế kiến trúc của chương trình ta chia ra từng Class, thì khi lập trình ta lại thiết kế chi tiết về Subs, Functions..., mỗi thứ sẽ cần phải thử như thế nào. Nếu ta có thể chia công việc ra từng giai đoạn thì mỗi giai đoạn có thể gọi đến một **Sub**. Thứ gì cần phải tính ra hay lấy từ nơi khác thì có thể được thực hiện bằng một **Function**.

Nhớ rằng điểm khác biệt chính giữa một Sub và một Function là Function cho ta một kết quả mà không làm thay đổi giá trị những tham số đầu vào mà ta cung cấp cho nó. Trong khi đó, dấu rằng Sub không cho ta gì một cách rõ ràng nhưng nó có thể thay đổi trị số (giá trị) của bất cứ tham số nào ta chuyển cho nó **ByRef**.

Do đó để tránh trường hợp vô tình làm cho trị số một biến bị thay đổi vì ta dùng nó trong một Sub/Function chúng ta nên dùng ByVal khi chuyển nó như một tham số vào một Sub/Function.

Thật ra, chúng ta có thể dùng ByRef cho một tham số chuyển vào một Function. Trong trường hợp đó dĩ nhiên biến ấy có thể bị sửa đổi. Điều này gọi là phản ứng phụ (**side effect**), vì bình thường ít ai làm vậy. Do đó, nếu chúng ta thật sự muốn vượt ngoài qui ước thông thường thì nên Comment rõ ràng để cảnh báo người sẽ đọc chương trình chúng ta sau này.

Ngoài ra, mỗi lập trình viên thường có một **Source Code Library** của những Subs/Functions ưng ý. Chúng ta nên dùng các Subs/Functions trong Library của chúng ta càng nhiều càng tốt, vì chúng đã được thử nghiệm rồi.

## II. Một số lưu ý

### 1. Dùng sợ lỗi

Mỗi khi chương trình có một Lỗi, hoặc là **Compilation Error** (vì ta viết code không đúng văn phạm, ngữ vựng), hoặc là lỗi trong khi chạy chương trình, thì chúng ta không nên sợ nó. Hãy bình tĩnh đọc cái **Error Message** để xem nó muốn nói gì. Nếu không hiểu ngay thì đọc đi đọc lại vài lần và suy nghiệm xem có tìm được sự hướng dẫn nào không. Khi lập trình chúng ta sẽ gặp lỗi rất nhiều, nên chúng ta phải tập bình tĩnh đối diện với chúng.



## 2. Dùng Comment (Chú thích)

Lúc viết code nhớ thêm Comment đầy đủ để bất cứ khi nào trở lại đọc đoạn code ấy trong tương lai chúng ta không cần phải dựa vào tài liệu nào khác mà có thể hiểu ngay lập tức mục đích của một Sub/Function hay đoạn code.

Như thế không nhất thiết chúng ta phải viết rất nhiều Comment nhưng hãy có điểm nào khác thường, bí hiểm thì chúng ta cần thông báo và giải thích tại sao chúng ta làm cách ấy. Có thể sau này ta khám phá ra đoạn code có bugs; lúc đọc lại có thể ta sẽ thấy đầu rằng ý định và thiết kế đúng nhưng cách lập trình có phần thiếu kiểm soát chẳng hạn.

Tính ra trung bình một lập trình viên chỉ làm việc 18 tháng ở mỗi chỗ. Tức là, gần như chắc chắn code chúng ta viết sẽ được người khác đọc và bảo trì ( debug và thêm bớt). Do đó, code phải càng đơn giản, dễ hiểu càng tốt. Đừng lo ngại là chương trình sẽ chạy chậm hay chiếm nhiều bộ nhớ, vì ngày nay computer chạy rất nhanh và bộ nhớ rất rẻ. Khi nào ta thật sự cần phải quan tâm về vận tốc và bộ nhớ thì điều đó cần được thiết kế cẩn thận chứ không phải dựa vào những tiêu xảo về lập trình.

## 3. Đặt tên các biến có ý nghĩa

Trong thực tế chúng ta gặp rất nhiều khó khăn khi làm việc với các biến có tên vắn tắt như K, L, AA, XY. Ta không có một chút ý niệm gì về chúng, mục đích sử dụng chúng làm gì. Thay vào đó, nếu ta đặt các tên biến như NumberOfItems, PricePerUnit, Discount .v.v.. thì sẽ dễ hiểu hơn.

Một trong những bugs khó thấy nhất là ta dùng cùng một tên cho **biến cục bộ** (biến được khai báo bên trong Sub/Function) và **biến toàn cục** (biến được khai báo trong Form hay Basic Module). Biến cục bộ sẽ che đậy biến toàn cục cùng tên, nên nếu chúng ta muốn nói đến biến toàn cục trong hoàn cảnh ấy chúng ta sẽ dùng làm biến cục bộ.

## 4. Dùng Option Explicit

Chúng ta nên trung thành với cách dùng **Option Explicit** ở đầu mỗi Form, Class hay Module. Nếu có biến nào đánh vắn sai VB6 IDE sẽ cho chúng ta biết ngay. Nếu chúng ta không dùng Option Explicit, một biến đánh vắn sai được xem như một biến mới với giá trị 0 hay "" (chuỗi rỗng).

Nói chung chúng ta nên thận trọng khi chỉ định một kiểu dữ liệu cho một biến với kiểu dữ liệu khác. Chúng ta phải biết rõ chúng ta đang làm gì để khỏi bị phản ứng phụ (side effect).

## 5. Desk Check

Kiểm tra lại chương trình trước khi biên dịch. Khi ta biên dịch mã nguồn chương trình, nếu không có lỗi chỉ có nghĩa là cú pháp của chương trình đúng, không có nghĩa là giải thuật đúng. Do đó ta cần phải biết chắc là mã chương trình ta viết sẽ làm đúng điều ta muốn bằng cách đọc lại chương trình trước khi biên dịch nó lần đầu tiên. Công việc này gọi là **Desk Check** (Kiểm tra trên bàn). Một chương trình được Desk Checked kỹ sẽ cần ít debug và chứa ít bugs không ngờ trước. Lý do là mọi kịch bản đã được tiên liệu chu đáo.

## 6. Soạn một Test Plan

Test Plan (kế hoạch kiểm thử) liệt kê tất cả những gì ta muốn thử và cách thử chúng. Khi thử theo Test Plan ta sẽ khám phá ra những bug và tìm cách loại chúng ra. Hồ sơ ghi lại lịch sử của Test Plan (trục trặc gì xảy ra, chúng ta đã dùng biện pháp nào để giải quyết) sẽ bổ ích trên nhiều phương diện. Ta sẽ học được từ kinh nghiệm Debug và biết rõ những thứ gì trong dự án đã được thử theo cách nào.

### III. Các kỹ thuật xử lý lỗi

#### 1. Xử lý lỗi lúc thực thi chương trình

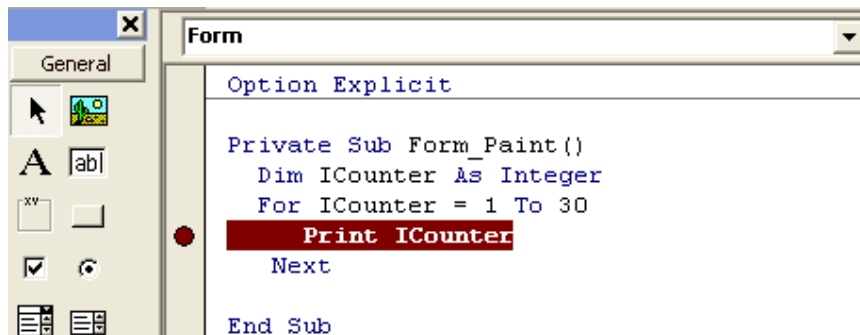
Khi một chương trình (đã dịch sang EXE) viết bằng VB6 đang chạy, nếu gặp lỗi, nó sẽ hiển thị một thông báo lỗi cho biết lý do gây lỗi. Sau khi chúng ta chọn OK, chương trình sẽ ngưng. Nếu chúng ta chạy chương trình trong VB6 IDE, chúng ta có dịp bảo chương trình ngưng ở trong mã nguồn chỗ có lỗi bằng cách bấm nút Debug trong thông báo lỗi. Tiếp theo đó chúng ta có thể tìm hiểu trị số các biến để đoán nguyên do của lỗi. Do đó, nếu chúng ta bắt đầu cho dùng một chương trình chúng ta viết cho nội bộ đơn vị, nếu tiện thì trong vài tuần đầu, thay vì chạy EXE của chương trình, chúng ta chạy source code trong VB6 IDE. Nếu có bug nào xảy ra, chúng ta có thể cho chương trình ngưng trong source code để debug.

Khi chúng ta dùng lệnh: *ON Error Resume Next* thì từ chỗ đó trở đi, nếu chương trình gặp lỗi, nó sẽ bỏ qua hoàn toàn. Điểm này tiện ở chỗ giúp chương trình EXE của ta tránh bị treo ngay lập tức tại điểm xuất hiện bug. Nhưng nó cũng bất lợi là khi khách hàng cho hay họ gặp những trường hợp lạ, không giải thích được (vì lỗi bị bỏ qua mà không ai để ý), thì ta cũng không biết được nguyên nhân, có thể không biết bắt đầu từ đâu để debug. Do đó, dĩ nhiên trong lúc debug ta không nên dùng nó, nhưng trước khi giao cho khách hàng chúng ta nên cân nhắc kỹ trước khi dùng.

#### 2. Dùng Breakpoints

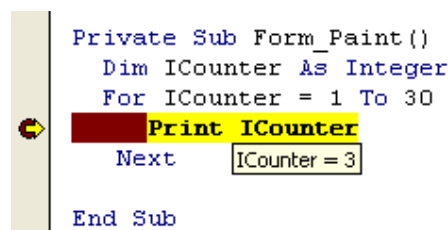
Cách hay nhất để theo dõi việc thực hiện của chương trình là dùng Breakpoint để làm cho chương trình ngưng lại ở một chỗ ta muốn ở trong chương trình, rồi sau đó ta cho chương trình bước từng bước. Trong dịp này ta sẽ xem xét trị số của những biến để coi chúng có đúng như dự định không.

Chúng ta đoán trước máy tính sẽ thực hiện các đoạn chương trình sẽ thực hiện và chọn một chỗ thích hợp rồi click bên trái của dòng code, chỗ dấu chấm tròn đỏ như trong hình dưới đây:

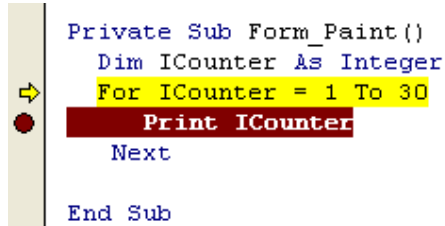


Nếu chúng ta click lên dấu chấm tròn đỏ một lần nữa thì là hủy bỏ nó. Một cách khác để đặt một breakpoint là để editor cursor lên dòng code rồi bấm **F9**. Nếu chúng ta bấm F9 lần nữa khi con trỏ nằm trên dòng đó thì là hủy bỏ breakpoint.

Lúc chương trình đang dừng lại, chúng ta có thể xem trị số của một biến bằng cách để con trỏ lên trên biến ấy, tooltip sẽ hiện ra như trong hình dưới đây:



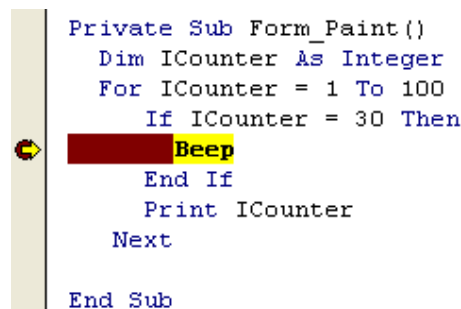
Có một số chuyện khác chúng ta có thể làm trong lúc này. Chúng ta có thể nắm dấu chấm tròn đồ kéo (drag) nó ngược lên một hay nhiều dòng code để nó sẽ thực thi trở lại vài dòng code. Chúng ta cho chương trình thực thi từng dòng code bằng cách bấm **F8**. Menu command tương đương với nó là **Debug | Step Into**. Sẽ có lúc chúng ta không muốn chương trình bước vào bên trong một Sub/Function mà muốn việc thực thi một Sub/Function như một bước đơn giản. Trong trường hợp đó, chúng ta dùng Menu command **Debug | Step Over** hay **Shift-F8**.



```
Private Sub Form_Paint()
    Dim ICounter As Integer
    For ICounter = 1 To 30
        Print ICounter
    Next
End Sub
```

Nhớ là để cho chương trình chạy lại chúng ta bấm **F5**, tương đương với Menu command **Run | Continue**.

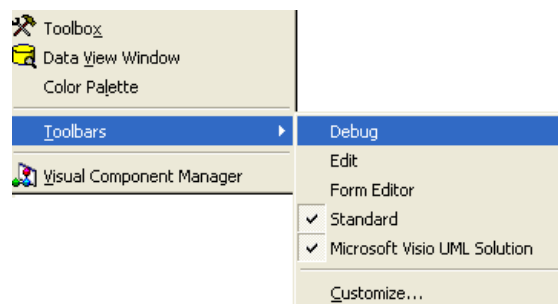
Có khi chúng ta muốn chương trình ngừng ở giữa một For Loop khi Iterator value có một trị số khá lớn. Nếu ta để sẵn một breakpoint ở đó rồi cứ bấm F5 nhiều lần thì hơi bất tiện. Có một phương pháp hữu hiệu là dùng một lệnh IF để thử khi Iterator value có trị số ấy thì ta ngừng ở breakpoint tại lệnh **Beep** (thay vì lệnh **Print ICounter**) như trong hình dưới đây:



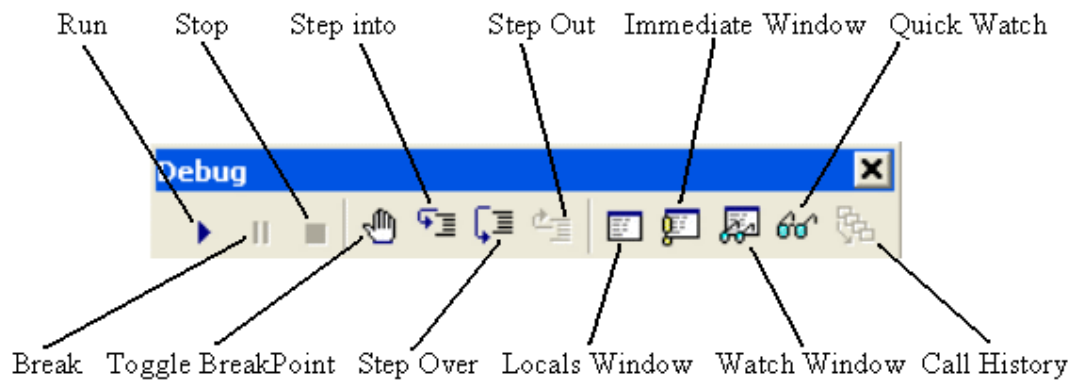
```
Private Sub Form_Paint()
    Dim ICounter As Integer
    For ICounter = 1 To 100
        If ICounter = 30 Then
            Beep
        End If
        Print ICounter
    Next
End Sub
```

Muốn hủy bỏ mọi breakpoints chúng ta dùng Menu command **Debug | Clear All Breakpoints**.

Để tiện việc debug, chúng ta có thể dùng **Debug Toolbar** bằng cách hiển thị nó với Menu command **View | Toolbars | Debug**



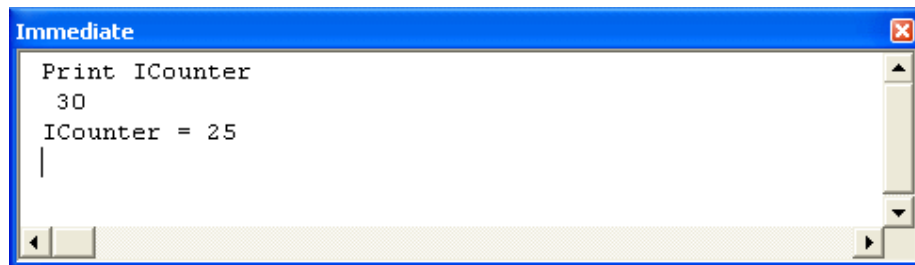
VB6 IDE sẽ hiển thị Debug Toolbar như sau:



### 3. Dừng Immediate Window

**Immediate Window** cho phép ta thực thi những lệnh VB trong khi chương trình đang dừng lại. Ta có thể dùng một lệnh Print để hiển thị trị số của một biến hay kết quả của một Function, gọi một Sub hay thay đổi trị số một biến trước khi tiếp tục cho chương trình chạy lại.

Để hiển thị Immediate Window, dùng Menu command **View | Immediate Window**.



Thay vì đánh "**Print ICounter**" chúng ta cũng có thể đánh "**? ICounter**". Nhớ là mỗi lệnh VB chúng ta đánh trong Immediate Window sẽ được executed ngay khi chúng ta bấm **Enter**. Chúng ta có thể dùng lại bất cứ lệnh VB nào trong Immediate Window, chỉ cần bấm Enter ở cuối dòng ấy.

### 4. Theo dấu vết chương trình (Tracing)

Đôi khi không tiện để ngừng chương trình nhưng chúng ta vẫn muốn biết chương trình đang làm gì trong một Sub. Chúng ta có thể để giữa code của một Sub/Function một lệnh giống như dưới đây.

Debug.Print Format ( Now,"hh:mm:ss ") & "(Sub ProcessInput) Current Status:" & Status để chương trình hiển thị trong Immediate Window value của Status khi nó thực thi bên trong Sub ProcessInput lúc mấy giờ.

Có một cách khác là thay vì cho hiển thị trong Immediate Window chúng ta cho viết xuống (**Log**) vào trong một text file. Dưới đây là một Sub điển hình chúng ta có thể dùng để Log một Event message:

```
Sub LogEvent(ByVal GivenFileName, ByVal Msg As String, HasFolder
As Boolean, IncludeTimeDate As Integer)
    ' Append event message Msg to a text Logfile GivenFileName
    ' If GivenFileName is fullPathName then HasFolder is true
    ' IncludeTimeDate = 0 : No Time or Date
    '   = 1 : Prefix with Time
    '   = 2 : Prefix with Time and Date
```

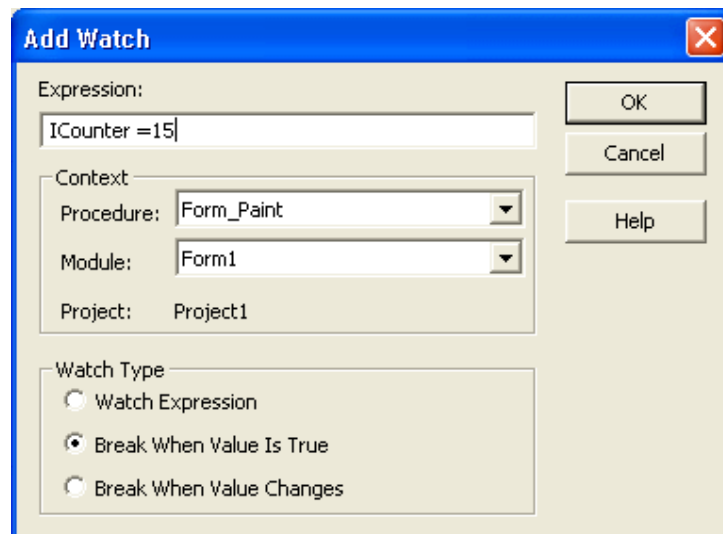
```

Dim FileNo, LogFileName, theFolder
If HasFolder Then
    LogFileName = GivenFileName
Else
    If Right(App.Path, 1) <> "\" Then
        theFolder = App.Path & "\"
    Else
        theFolder = App.Path
    End If
    LogFileName = theFolder & GivenFileName
End If
FileNo = FreeFile
If Dir(LogFileName) <> "" Then
    Open LogFileName For Append As FileNo
Else
    Open LogFileName For Output As FileNo
End If
Select Case IncludeTimeDate
Case 0 ' No Time or Date
    Print #FileNo, Msg
Case 1 ' Time only
    Print #FileNo, Format(Now, "hh:nn:ss ") & Msg
Case 2 ' Date & Time
    Print #FileNo, Format(Now, "dd/mm/yyyy hh:nn:ss ") & Msg
End Select
Close FileNo
End Sub

```

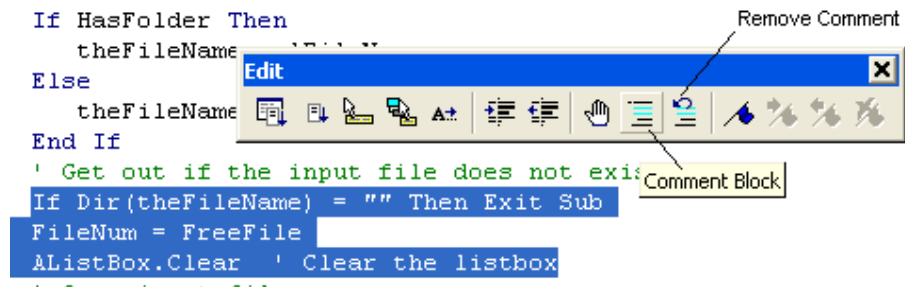
## 5. Dừng Watch Window

Đôi khi chúng ta muốn chương trình ngừng không phải ở một chỗ nào nhất định, nhưng khi trị số của một biến hay của một biểu thức là bao nhiêu, có thể là chúng ta không biết tại sao một biến tự nhiên có một trị số như vậy. Ví dụ, chúng ta muốn chương trình ngừng lại khi **ICounter = 15**. Chúng ta có thể dùng Menu command **Debug | Add Watch**. VB6 IDE sẽ hiển thị hộp thoại dưới đây. Chúng ta đánh **ICounter = 15** vào textbox **Expression** và click option box **Break When Value Is True** trong hộp **Watch Type**. Làm như vậy có nghĩa là ta muốn chương trình ngừng khi ICounter bằng 15.



## 6. Dùng phương pháp loại suy (Elimination Method)

Có một phương pháp rất thông dụng khi debug là loại bỏ những dòng code nghi ngờ để xem bug có biến mất không. Nó được gọi là **Elimination Method**. Nếu bug biến mất thì những dòng code đã được loại bỏ là thủ phạm. Chúng ta có thể Comment Out một số dòng cùng một lúc bằng cách highlight các dòng ấy rồi click **Comment Block** trên Edit ToolBar.



Khi dùng Elimination Method chúng ta phải cân nhắc thuật toán của chương trình chúng ta trong khi quyết định Comment Out những dòng nào, nếu không, đó là một phương pháp khá nguy hiểm.

Ngoài ra, Menu Command **View | Locals Window** liệt kê cho chúng ta trị số của tất cả biến trong một Sub/Function và **View | Call Stack** liệt kê thứ bậc các Sub gọi lần lượt từ ngoài vào trong cho đến vị trí code đang ngừng hiện thời.

# TÀI LIỆU THAM KHẢO

- [1] Brown M. and Sedgewick R., *A system for algorithm animation*. In Proc. Of SIGGRAPH '84, pp. 177–186, 1984.
- [2] Burnett M. and Ambler A. L., *A declarative approach to event-handling in visual programming languages*. In Proc. 1993 IEEE Symposium Visual Languages, pp. 34–40, Seattle, Washington, September 1992.
- [3] Burnett M., and Baker M. J., *A classification system for visual programming languages*. J. Visual Languages and Computing, pp. 287–300, September 1994.
- [4] Chang S., *Visual languages: A tutorial and survey*. IEEE Software, 4(1):29–39, January 1987.
- [5] Chang S.-K., *Principles of Visual Programming Systems*. Prentice Hall, New York, 1990.
- [6] Cox P. T. and Pietryzkowsky T., *Using a pictorial representation to combine dataflow and object-orientation in a language-independent programming mechanism*. IEEE Computer Society Press, Los Alamitos, CA, 1990.
- [7] Erwig M. and Meyer B., *Heterogeneous visual languages : Integrating visual and textual programming*. In Proc. 1995 IEEE Symposium Visual Languages, pp. 318–325, 1995.
- [8] Golin E. J., *A method for the specification and parsing of visual languages*. PhD dissertation, Brown University, 1990.
- [9] Lakin F., *Spatial parsing for visual languages*. In Chang, S.-K., Ichikawa, T., and Ligomenides, P., editors, Visual Languages, pp. 35–85. Plenum Press, New York, 1986.
- [10] Najork M., *Visual programming in 3-d*. Dr. Dobb's Journal, 20(12):18–31, December 1995.
- [11] Litwin P., Getz K., Mike Gilbert, *Access 2000 Developer's Handbook*. Published by Sybex, ISBN-10: 0782123708, September 1999.
- [12] Rod S., *Visual Basic 2005 Programmer's Reference*. Published by Wrox, ISBN: 978-0-7645-7198-5, October 2005.

# MỤC LỤC

<b>LỜI NÓI ĐẦU .....</b>	<b>3</b>
<b>CHƯƠNG 1. LẬP TRÌNH TRỰC QUAN.....</b>	<b>5</b>
<b>I. Giới thiệu .....</b>	<b>5</b>
<b>II. Lịch sử của các ngôn ngữ lập trình trực quan.....</b>	<b>5</b>
<b>III. Phân loại các ngôn ngữ lập trình trực quan .....</b>	<b>6</b>
<b>IV. Lý thuyết của các ngôn ngữ lập trình trực quan.....</b>	<b>8</b>
1. Đặc tả hình thức của ngôn ngữ lập trình trực quan .....	8
2. Phân tích các ngôn ngữ lập trình trực quan .....	10
<b>V. Những vấn đề của ngôn ngữ trực quan .....</b>	<b>11</b>
1. Control Flow (luồng điều khiển) .....	11
2. Sự trừu tượng hoá thủ tục (Procedural Abstraction) .....	11
3. Sự trừu tượng hoá dữ liệu (Data Abstraction) .....	12
<b>VI. Các ngôn ngữ lập trình trực quan .....</b>	<b>12</b>
1. ARK .....	12
2. Prograph.....	14
3. Form/3.....	16
<b>VII. Kết luận .....</b>	<b>18</b>
<b>CHƯƠNG 2. LẬP TRÌNH TRỰC QUAN VỚI MS ACCESS .....</b>	<b>20</b>
<b>I. Giới thiệu .....</b>	<b>20</b>
1. Khái niệm về cơ sở dữ liệu .....	20
2. Microsoft Access .....	24
3. Khởi động ACCESS .....	25
4. Cơ sở dữ liệu trong Access .....	25
5. Các phép toán.....	25
<b>II. Làm việc với cơ sở dữ liệu (CSDL) .....</b>	<b>26</b>
1. Tạo cơ sở dữ liệu .....	26
2. Hiệu chỉnh cơ sở dữ liệu .....	27
<b>III. Làm việc với Table .....</b>	<b>28</b>
1. Tạo cấu trúc của Table.....	28
2. Nhập số liệu vào Table .....	30
3. Hiệu chỉnh Table.....	31
4. Khai thác số liệu trên Table .....	32
<b>IV. LÀM VIỆC VỚI QUERY .....</b>	<b>32</b>
1. Khái niệm.....	32
2. Cách tạo QUERY.....	33
3. Hiệu chỉnh QUERY .....	37
4. Thực hiện QUERY .....	37
<b>V. Làm việc với Report .....</b>	<b>37</b>
1. Khái niệm.....	37
2. Cách tạo Report .....	38



3. Hiệu chỉnh Report .....	41
4. Thực hiện Report.....	41
<b>VI. Làm việc với Form.....</b>	<b>41</b>
1. Khái niệm .....	41
2. Thiết kế Form.....	41
3. Hiệu chỉnh Form .....	43
4. Thực hiện Form.....	43
<b>VII. Macro và hệ thống thực đơn .....</b>	<b>43</b>
1. Macro .....	43
2. Hệ thống thực đơn.....	44
<b>CHƯƠNG 3. BẮT ĐẦU LẬP TRÌNH VỚI VISUAL BASIC .....</b>	<b>47</b>
<b>I. Giới thiệu .....</b>	<b>47</b>
<b>II. Các khái niệm thường dùng.....</b>	<b>48</b>
<b>III. Lập trình trong Visual Basic.....</b>	<b>48</b>
1. Làm việc với hộp/nút điều khiển .....	49
2. Thuộc tính .....	51
3. Thủ tục tình huống .....	53
<b>IV. Ví dụ .....</b>	<b>54</b>
1. Bổ sung hộp điều khiển.....	54
2. Thay đổi thuộc tính .....	54
3. Viết các thủ tục tình huống .....	55
4. Ghi và thực hiện trương trình.....	56
<b>V. Biến nhớ .....</b>	<b>61</b>
1. Khái niệm.....	61
2. Khai báo biến .....	61
3. Khai báo hằng .....	63
4. Mảng .....	63
5. Khai báo bản ghi .....	64
6. Biến đổi (convert) từ loại dữ liệu này qua loại dữ liệu khác .....	64
<b>CHƯƠNG 4. VIẾT MÃ CHƯƠNG TRÌNH.....</b>	<b>66</b>
<b>I. Các cấu trúc điều khiển.....</b>	<b>66</b>
1. Cấu trúc chọn .....	66
2. Cấu trúc lặp .....	67
3. Nhãn .....	69
4. Số thứ tự dòng lệnh .....	70
<b>II. Method .....</b>	<b>70</b>
1. Circle Method.....	70
2. Line Method .....	71
3. Cls Method.....	72
4. Hide Method.....	72
5. Show Method .....	73
6. Item Method .....	73
7. Move Method.....	73
8. Point Method.....	74

9. Print Method .....	74
10. PrintForm Method.....	74
11. PSet Method.....	75
12. Refresh Method.....	75
13. Scale Method .....	76
14. SetFocus Method .....	77
15. TextHeight và TextWidth Methods .....	77
<b>III. Hàm.....</b>	<b>78</b>
1. Giới thiệu .....	78
2. Các hàm xử lý chuỗi .....	78
3. Các hàm xử lý số : .....	80
<b>CHƯƠNG 5. THIẾT KẾ GIAO DIỆN .....</b>	<b>81</b>
<b>I. Giới thiệu .....</b>	<b>81</b>
<b>II. Dùng list control .....</b>	<b>81</b>
1. Listbox .....	82
2. Drag-Drop .....	85
3. Dùng thuộc tính Sorted .....	86
<b>III. Tự tạo các đối tượng (Object) .....</b>	<b>89</b>
<b>CHƯƠNG 6. CÁC CHẾ ĐỘ HỘI THOẠI .....</b>	<b>97</b>
<b>I. Message Boxes (hộp thông điệp) .....</b>	<b>97</b>
<b>II. Input Boxes.....</b>	<b>99</b>
<b>III. Common Dialogs.....</b>	<b>101</b>
<b>IV. Open và Save File Dialogs.....</b>	<b>101</b>
<b>V. Các loại Dialog có sẵn để dùng.....</b>	<b>105</b>
1. Color Dialog .....	105
2. Font Dialog .....	107
3. Print Dialog.....	109
4. Help Dialog.....	110
<b>VI. Custom Dialogs .....</b>	<b>110</b>
<b>CHƯƠNG 7. THIẾT KẾ ĐỒ HỌA.....</b>	<b>113</b>
<b>I. Màu (color) và độ phân giải (resolution).....</b>	<b>113</b>
1. Độ phân giải (resolution) .....	113
2. Màu (color) .....	114
<b>II. Hàm RGB .....</b>	<b>116</b>
<b>III. Một số kỹ thuật .....</b>	<b>118</b>
1. Color Mapping .....	118
2. Dùng Intrinsic Color Constants .....	119
3. Tập tin đồ họa .....	120
<b>CHƯƠNG 8. KẾT NỐI ĐẾN CƠ SỞ DỮ LIỆU .....</b>	<b>121</b>
<b>I. Sử dụng Data Control .....</b>	<b>121</b>

1. Chỉ định vị trí cơ sở dữ liệu lúc chạy chương trình .....	124
2. Thêm bớt các bản ghi .....	125
3. Dùng DataBound Combo .....	127
<b>II. Lập trình với kỹ thuật DAO .....</b>	<b>129</b>
1. Tham chiếu DAO .....	129
2. Dùng keyword SET .....	130
3. Các nút di chuyển .....	131
4. Thêm bớt các bản ghi .....	132
5. Tìm một bản ghi .....	134
6. Bookmark .....	137
7. LastModified .....	137
<b>III. Lập trình với ADO.....</b>	<b>138</b>
<b>IV. Data Form Wizard.....</b>	<b>142</b>
 <b>CHƯƠNG 9. THIẾT KẾ HỆ THỐNG THỰC ĐƠN .....</b>	 <b>149</b>
<b>I. Main Menu (Thực đơn chính) .....</b>	<b>149</b>
<b>II. Pop-up Menu .....</b>	<b>153</b>
<b>III. Lưu trữ việc thiết lập thực đơn trong Registry.....</b>	<b>153</b>
 <b>CHƯƠNG 10. GỠ LỖI CHƯƠNG TRÌNH.....</b>	 <b>159</b>
<b>I. Đặc tả chương trình (Program Specifications) .....</b>	<b>159</b>
1. Cấu trúc các bộ phận .....	159
2. Kỹ thuật lập trình .....	160
3. Dùng Subs và Functions .....	160
<b>II. Một số lưu ý .....</b>	<b>160</b>
1. Dùng sơ lỗi .....	160
2. Dùng Comment (Chú thích).....	161
3. Đặt tên các biến có ý nghĩa .....	161
4. Dùng Option Explicit .....	161
5. Desk Check .....	161
6. Soạn một Test Plan.....	161
<b>III. Các kỹ thuật xử lý lỗi.....</b>	<b>162</b>
1. Xử lý lỗi lúc thực thi chương trình.....	162
2. Dùng Breakpoints.....	162
3. Dùng Immediate Window .....	164
4. Theo dấu vết chương trình (Tracing) .....	164
5. Dùng Watch Window.....	165
6. Dùng phương pháp loại suy (Elimination Method) .....	166
 <b>TÀI LIỆU THAM KHẢO .....</b>	 <b>167</b>
 <b>MỤC LỤC.....</b>	 <b>168</b>